UNIVERSITY OF OSLO
Department of Informatics

If your Webserver was an Animal, how would you describe it?

Johan Finstadsveen

24.May 2011

If your Webserver was an Animal

How would you describe it?

Johan Finstadsveen

July 10, 2011

Abstract

This master thesis demonstrates how biological mechanisms and animal behavior can be used to reduce the conceptual complexity of automated processes in virtual machines. By translating different types of behavior and mechanisms into automated management processes the goal is to reduce the need for management of virtual machines with regards to resources, reliability and security.

Acknowledgements

First and foremost I would like to thank Kyrre Begnum for his dedication, enthusiasm and support. With his guidance, I have been able to create a project that has been interesting and a joy to work with from beginning till end. His ideas and visions have helped formed this project, and for this I will be forever grateful.

Secondly I would like to extend my gratitude to Oslo University College and the University of Oslo for making this master program possible. However a house alone does not make a home, so this masters would be nowhere without the dedication, commitment and skills of teachers such as Hårek Haugerud, Siri Fagernes, Simen Hagen and Æleen Frisch.

I would also like to thank all my classmates, family and friends for their continuous support and understanding during this masters. For all the late night advices, the moral support and words of encouragement, technical help and the trips to Evergreen, for this I thank all of you.

Finally I would like to thank Anette for her everlasting patience and positive encouragement. I would not be able to do this without you.

Contents

Chapter 1

Introduction

Managing computer systems are getting more complex. Updates, upgrades, deployments, migrations and scaling are just some tasks that system administrators are expected to do. While doing this, they are expected to use as little time and resources as possible whilst maintaing a secure and efficient computer system.

A part of this added complexity is the ever increasing popularity of virtual machines. Virtual machines are changing the way computer infrastructure are deployed and managed. In addition to its benefits, virtual machines also pose great challenges that need to be addressed in order for it to be a sustainable platform. The future of virtual machines depends on how the platform can manage the increased complexity and size of the increasing number of virtual machines and services.

More virtual machines will be deployed in 2011 than in the period 2001-2009 put together [33]. The research firm Gartner estimates that by 2013 the majority of the workload running on x86 architecture is running on virtual machines. The increasing number of virtual machines will increase the complexity of management, which again makes it more difficult to manage the resources in an effective way. This could lead to poor return of investment(ROI), making the shift to virtual machines less effective.

The challenge of an increasing virtual environment is not the only complicating factor in a large virtual deployment scenario. The virtual architecture should allow scaling, and the scaling should be able to dynamically allocate the correct amount of resources. Virtual machines also offer a possibility of host migration which enables several new scenarios and further complicates the management. Manual scaling and migration of hosts takes up time for a system administrator, resources which could be more effectively used at other tasks. These are new types of operations that will be included into the

existing service and systems management paradigm.

The need and desire for automating manual simple tasks in system administration is apparent. Humans are in many ways not ideal for managing dynamic systems. System administrators cannot work day and night, they constantly need bathroom breaks, lunch and vacations. This will leave gaps in a decision making system, thus not allowing a dynamical scaling of resources. A system that monitors the resources and autonomic makes changes to adapt to new situations without the need for a system administrators manual decision making or input is needed.

Different approaches for creating management solutions exists, but are often far too complicated for a system administrator to implement. This is either due to the actual technical nature of the model or the lack of profession-oriented research towards system administration. A new model must not only work as a solution, but it must also be presented in such a way the users and implementors will understand. This means covering the technical aspects of the solution, but also their function. However, the function must be explained using metaphors and analogies that is easier understood.

There is a need of simplifying the concepts of system monitoring and autonomic management. By simplifying complicated management processes one can easier facilitate the threshold for system administrators to implement and use a complex solution.

Managing the resources in a virtual machine in an effective manner whilst still maintaing the advantages of virtualization is therefore a key factor to have a successful deployment scenario. The beneficiary of increasing the effectiveness of the resource-usage, but also minimizing the amount of manual management could have major effect on the ROI.

Different approaches to managing virtual resources exits, but is often an additional product such as VMWare Dynamic Resource Scheduler[63]. One of issues with these types of solutions is that it is an additional product that must be purchased, which will increase the cost of a virtual environment. The product is also proprietary which allows for no possible modification of the software. The need for a free and open source alternative for effective resource management of virtual machines is necessary.

Biological methods have previously been used in other computer fields. Fred Cohen[18] used the biological term "Virus" to explain a computer phenomena, but he also created an analogy that took a hugely complicated scenario and made it easily understandable by using terms already known. Creating a level of abstraction creates terms that can be explained to people that do not

have a computer background. Today terms like worms and viruses are common for computer threats. However, Cohens approach used the term from biology, but did not use knowledge from biology to do actual implementation.

By using both terms and knowledge from the field of biology one could both explain a difficult term by a proper analogy, but one could also re-use the knowledge and methods found. Biological methods and strategies exists and are well documented. As an example of Biology and Computer Science collaboration, the human immune system has been used to create both analogies and real-life implementations[14][29]. Mark Burgess demonstrates how different programs could work together towards creating a computer immune system.

Developing software is expensive, time-consuming and demands a continuous cycle of updates to support different platforms. A solution based on orchestrating, meaning several different solutions working together for a common goal, is easier to create. The task of supporting and updating the tool to work with updates and newer versions will fall to that of the developer of the initial program. This allows for a more sustainable approach and should thus be more prone to be adapted by system administrators.

The importance of an approach that is comprehensive and addresses more than a single subject is needed. The implementations so far have had a focus on a single subject, for example one for security, one for reliability and another solution for resources. This adds another level of complexity and creates a barrier between the software and the system administrator that is implementing the solution. The need for a comprehensive solution is apparent. Adapting biological methods to manage the clouds resource and security makes for an interesting new approach, with focus on reusing old knowledge to a new problem

1.1 Problem statements

Q1 - How can biological mechanisms and animal behavior be used to simplify the conceptual complexity of automated processes in virtual machines.

Q2 - How can biological mechanisms and animal behavior be translated into automated management processes of virtual machines with regards to:

- Resources
- Security

- Reliability

The biological definition of behavior can be explained by psychologists B.F Skinner and D.O. Hebb. "All observable processes by which an animal responds to perceived changes in the internal state of its body or in the external world"[7]. It also covers other types of non-visual behavior such as cognition processes and motivation. Mechanisms in this context covers how the desired behavior is achieved by the animal.

The term biological is limited to only include animals and not for example plants. Examples of biological behavior or mechanisms can be social groups in the animal world, also known as herds. Working as a group, a stalking predator is likely to be detected earlier and thus making it possible for the herd to choose to fight or flight. By using the size of the heard as an advantage, the animals ensure the survival of both genes and species.

Animals is not only restricted to mammals, but also includes insects, birds and reptiles among others. Predominantly the term will be used on wild animals in their natural habitat. It will not describe typical domestic house animals such as cats and dogs and their unconventional behavior.

Mechanisms also covers the different methods animals uses to adapt, survive and reproduce in their natural environment. In this context, adaptations is how animals adapt to new threats, changing of resources such as food or the environment or other external factors. Mechanisms covers different methods animals use as protection, but mainly focuses on different methods of anti-predator protection.

By using biological terms and knowledge one can abstract complicated processes in automated virtual machine management. Analogies and metaphors can be used to explain complicated technical solutions, as previously done with the computer virus. Simplifying this technical knowledge is not only important in an education scenario, but can also be used for system administrators to better grasp the necessity of the different technical concepts.

Automated management processes covers how to manage virtual machines in an autonomic fashion. An autonomic system operates and serves its purpose by managing itself without external intervention, i.e. a system administrators intervention. The system does so even in the case of an environmental change. To limit this approach it is narrowed down to resources, security and reliability.

Resources covers consumption of computing resources in a virtual machine environment. This area spans from the scaling of infrastructure to cope with

changes in the required service, to the automatic mechanisms to reduce the amount of resources consumed.

Security covers the principle of computer security. Mainly it focuses on the CIA-model, confidentiality, availability and integrity. The goal is to create mechanisms that addresses one or more of these fields. The term security covers both individual server security and the service provided as a whole.

By introducing aging of virtual machines one can increase the reliability of the service. Poorly designed software can allocate more resources over time than needed. One possibility is to recover from software-flaws which may have a decremental effect on the reliability of the service over time. Using age to define a virtual machines state allows the management system to create new virtual machine to replace the older machines, thus increasing the reliability of the service.

The term virtual machines covers the use of virtualization technologies in data-centers or in the cloud that provides some sort of service. An example of this could be a web service. Using workstation utilizing virtualization technology for a single virtual computer does not fall under the datacenter terminology.

Table 1.1: Thesis summary table

Theme	Project	Approach	Results
Q1 **Reduce conceptual complexity**	Create methodology	Creating a method used	Teaching
	Demonstrate how to reduce complexity by using biology	Create models using method	Explain biological term, demonstrate how this can be transferred to system administration
	Formalization	Design model	Design model using BRIC, explain BRIC-model to highlight design choices
Q1 **Using biology to increase management automation**	Aging	Implementation and testing	Created a script which rotates machine, i.e. oldest servers die to improve reliabilty
	Herd size	Implementation	Introduce a script that dynamically changes the size of a herd (number of servers) depending on traffic
	Attack warning	Implementation	Introduce a mechanism that demonstrates the use of communication between hosts to warn of an incoming attack
	Metabolism	Implementation	Demonstrate how servers could increase and decrease in size depending on load to more effectively use resources
	Testing	Test implemented models against simulated real-life traffic	Demonstrate how the different models can be used to improve management of resources, security and reliability against recreated real-life traffic

Chapter 2

Background

This project consist of two different fields of science, Biology and System Administration. Primarily within system administration the focus area is virtualization and cloud computing, as well as automated management. Biology is a exceptionally large field covering everything from cells and proteins to animal behavior. This approach is however limited to the two fields animal behavior and mechanisms.

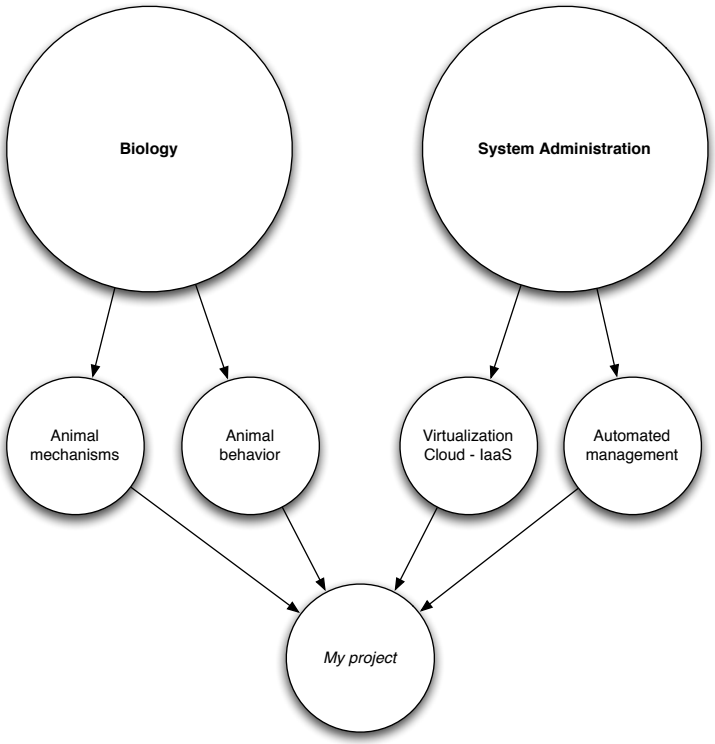


Figure 2.1: Fields of study

2.1 Virtual machines and IaaS

Virtual machines have been existing in various forms since the early IBM mainframes in the 1960s[20]. Virtualization makes it possible to have multiple operating systems on a single piece of hardware by sharing the hardware resources. This have several benefits in form of flexibility, higher server-utilization, increased security and more manageable management.

By sharing the resources on a physical server, less physical machines can be deployed. Virtualization has been a hyped of the green-wave in IT over the last years, where it has been presented as part of the solution to the vast amount of resources the IT-sector consumes[50][1]. More virtual machines will be deployed in 2011 than in the period 2001-2009 put together[33], so the derogatory term "hype" is clearly misused.

Virtualization also offers new types of functionality to that of the old paradigm. Migration[16] are one of these new functionalities. Migrating makes it possible to move virtual machines to different physical machines. This offers several new possibilities, amongst which prominent is fault management, load-balancing, and low-level system maintenance. Machines could simply be moved or additional machines can be started/stopped to cope with a dynamic load. This allows for a further increase in resource-utilization.

Virtualization was the stepping-stone that made cloud computing [5] and IaaS[60] a viable option. The possibility of dynamically scale to a theoretical infinite number of virtual servers and only pay for the hours used is a promising prospect. In practice however this as it's obvious limitations. The shift from a localized data center towards the cloud is one of the largest paradigm shifts in modern computing.

2.2 Platform

Amazon EC2, or "Amazon Elastic Compute Cloud"[8] was one of the first providers to deliver a cloud and have also been heavily involved in the development of the cloud. EC2 uses Xen virtualization to provide virtual machines, or instances as they are refereed to by Amazon.

Table 2.1: Prices for small Amazon EC2 instance(s) (19.may 2011)

	Prices in US dollars			
Number of servers	**Hour**	**Day**	**Month**	**Year**
1	0,03	0,72	21,6	262,8
5	0,15	3,6	108	1314
10	0,3	7,2	216	2.628

Xen is an open source virtual machine monitor originally developed by the University of Cambridge Computer Laboratory. It was made famous with the "Xen and the art of Virtualization" in 2004. [6].

MLN (Manage Large Networks) is a tool to build and run virtual machines based on Xen, Amazon EC2 and User-mode Linux as well as the former VMware Server. The goal of MLN is to have an easy way of configuring and managing large groups of virtual machines[9]. MLN allows the creation of images that can be uploaded to the Amazon EC2 cloud and management through MLN. MLN was created by Kyrre Begnum at Oslo University College.

MLN is continuously developed and is available freely at Sourceforge[10] as open-source and under the GPL-license. MLN is not widely deployed in production environments, but has been very successful in large scale education of system administration[44]. MLN has a plugin to simplify the the process of managing EC2 instance deployment and management[12].

2.2.1 Amazon SQS

Amazon SQS or Simple Queue Service is distributed queue message system provided by Amazon[3]. It allows sending messages through the internet by a pay per message method. Amazon SQS provides a high available authenticated message system similar to the Java Message Service. The design of Amazon SQS in such a way that there is no need for users to have dedicated servers for the messaging system. SQS allows to send messages up to 64KB in size. The messages have a latency pending between 2-10 seconds and can be stored up to 14 days. Pricing for SQS is dependent on two factors; data transferred and number of SQS requests.

The data transferred costs less than 0.150 US dollars per transferred GB (updated prices and region information is available at http://aws.amazon.com/sqs/). The cost of 10.000 SQS requests is 0.01 US dollars, or $0.000001 per request.

Perl and SQS

A perl module called Amazon SQS simple[65] allows the use of Perl to send and receive messages with SQS. It is freely available through CPAN. The module allows the creation and deletion of queues and messages. It also allows the setting of timeout periods for messages, thus allowing the messages to be sent to multiple users simultaneously.

Table 2.2: Communicating using Amazon SQS with a single server pricing example. (19.may 2011)

	Prices in US dollars				
Messsages/min	Per hour	Per day	Per Month	Per year	Messages - per year
60	0,0036	0,0864	2.592	31.104	31.104.000
4	0,00024	0,00576	0,1728	20.736	2.073.600
1	0,00006	0,00144	0,5184	51,84	518.400

Communicating every seconds is clearly costly and would amount to a vast number of messages (31.104.000 per machine). This would result in enormous log-files and large data-transfer fees. Switching to 15 second intervals gives a drop to 2.073.600 messages per year and is much cheaper. A per minute based message is very cheap, but does not offer the frequent updates that a dynamic service require.

Considering a scenario where a 100 servers are deployed the total cost of a 15-seconds scenario would be 207.36 US dollars per year. In comparison sending a message each second would amount to over 3110 US dollars in communication alone. A per minute would be as little as 51 US dollars. Considering the vast amount of messages sent from a large number of hosts the price should be regarded as acceptable. However the frequency of the updates should be set by the policy of each company/organization due to different needs.

2.3 Tools

The section presents a short outline and description of the tools being used in this thesis.

Snort

Snort is an open source network intrusion prevention and detection system (IDS/IPS) developed by Sourcefire[52]. It is a signature and anomaly based intrusion detection system, and allows for the creation of customized rules and alert levels. Snort was originally released in 1998, but is still being developed. It is available for both Linux and Windows.

Httperf

Httperf is a tool for measuring and benchmarking webserver performance[40][49]. It was created in 1998 by HP research labs and is available freely at sourceforge through the GNU GPL. Httperf is highly customizable allowing for easily scripting or changing the test settings in terms of load or number of requests.

2.4 BRIC

BRIC or block-like representation of interactive components is a high-level language for the design of multi-agent systems based on a modular approach[28]. A BRIC system consists of a set of components linked to each other by communication links.

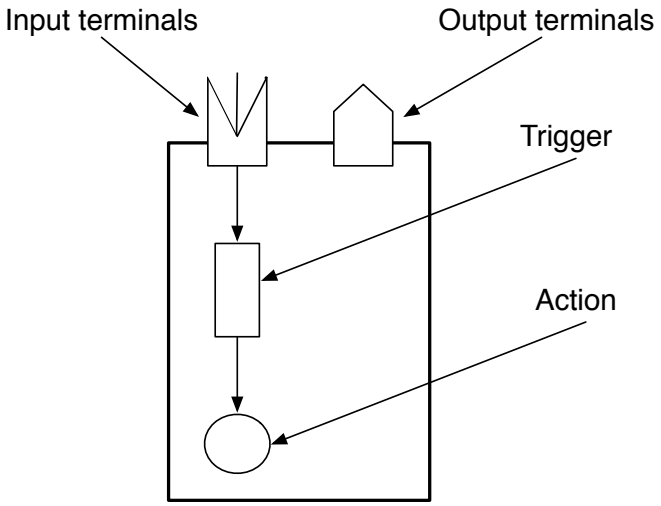


Figure 2.2: Explanation of BRIC components

The figure 2.3 demonstrates how the BRIC-model works. The different components receives messages through the input-terminals. Messages sent is demonstrated by the output terminal. This demonstrates the flow, dependencies and highlights the decision-making process.

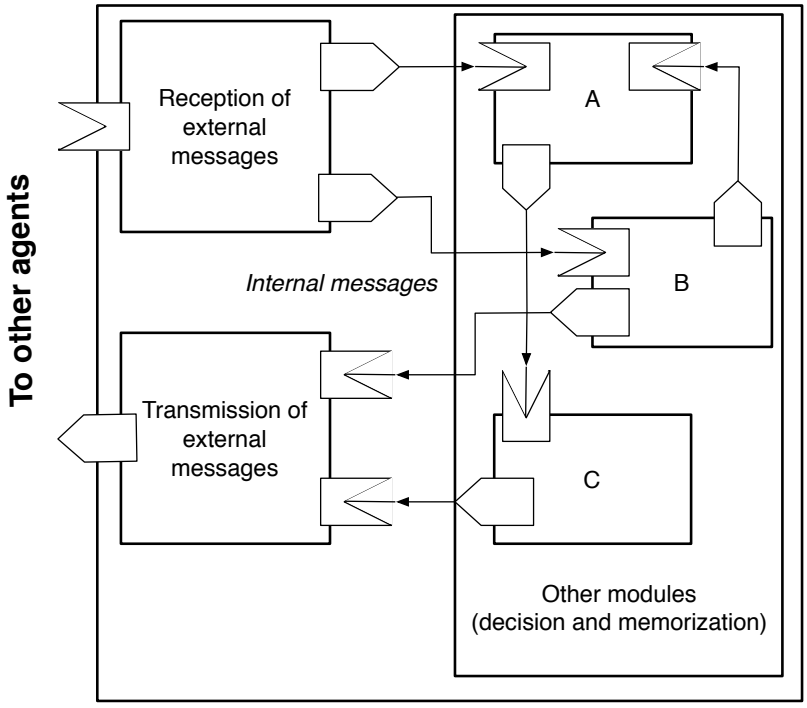


Figure 2.3: Example of BRIC-diagram

2.5 Using analogies

An analogy is a similarity between concepts. Although perhaps regarded as simplistic, using analogies is widely used in teaching. Using metaphors the presenter is able to present a bridge between the known and the new. This method is called "The Teaching With Analogies Model", and is an abstract model to explain a complex target concept[34].

Typically the proposed model uses the following steps:[25]

1. Introduce the target concept
2. Review the analog concept
3. Identify relevant features of the target and analog
4. Map similarities
5. Indicate where the analogy breaks down
6. Draw conclusions

This model is used in science education to help students understand complex, hard-to-visualize system with interacting parts. One of the reasons why the

analogy model is so successful is that listener get mental participation, they become part of the process and learning experience[46].

2.6 Biology

Biology is a natural science concerned with the study of life and living organisms, including their structure, function, growth, origin, evolution, distribution, and taxonomy. However the focus of this thesis is on the following subjects:

- Animal behavior
- Animal mechanims
  - Defense mechanisms

2.6.1 Animal behavior

Ethology or the the scientific study of animal behavior. It is sub-topic of Zoology which in terms is a branch of the large umbrella that is biology. In many aspects it is similar to comparative psychology[7], but has also strong relation with other disciplines such as neuroanatomy, ecology and evolution. The modern view on ethology is considered to have began in the 1930s by Nikolaas Tinbergen and Konrad Lorenz and Karl von Frisch[7].

2.6.2 Animal mechanisms

Most animals can be considered being predators of one sort or one another[7]. The diversity of animal life is sustained by the transfer of energy from one entity to one other. This could be plants, other animals, to even bacteria or fungi. To describe different types of animals and their source of income people use words like "herbivore - plant eater" and "carnivore - meat eater". Regardless of their source of income, the animal is faced with the same type of issues with regards to food and survival.

Given a evolutionary look at animal mechanisms, it is easy to see how different mutations could give an animal an advantage. Small mutations that may give an animal just a little advantage, may increase it chance to obtain food (Giraffes getting taller), may decrease it's chance to get detected (Chameleon camouflage) or avoiding consumption (Lizard loosing tale). Anti predator mechanisms can be behavioral, morphological or physiological.

The term reproduce when used in this thesis means how animals reproduce with regards how the size of the heard dynamically changes and not

the focus of reproduction in terms of coitus. The balance between prey and hunter would also be an interesting method to discuss. Mechanisms does not cover immune systems or other internal biological systems.

2.7 System Administration and Biology

Using biology has previously been used in system administration[14][29] to some extent. In the case of Forrest and the computer virus, a complicated computer phenomena was explained using biological reference. The term virus is still being used and understood by technical as well as non-technical personnel. Mark Burgess uses biology to explain a desired behavior or mechanism in system administration[14].

Reverse approach

The article "Using Robots to Understand Animal Behavior"[64] demonstrates how robots can be used to understand animal behavior. This article however focuses on how robotics can simplify the process of measuring stimuli, responses and behavior. This approach is however not widely used.

Chapter 3

Approach

Due to the nature of this assignment the approach must be divided into two different parts. The two different problem statements need separate and different approaches. The results from Q1 will form the basis of the approach for Q2. Q1 is therefore the first priority.

3.1 Q1 - A modeling approach

To simplify the conceptual complexity of different automated processes requires two parts, modeling and description. The model would break-down the different parts of the process and show the flow of a process. The description would be an attempt to explain a complex process with the use of animal behavior and mechanisms.

3.1.1 Animal behavior or mechanism as a description

The need for a scientific approach towards animal behavior and mechanisms is required. The knowledge of animals are well documented through the field of biology, so the documentation is available. "Birds fly" is a general statement that everyone knows it's true. However, if to be used as a starting point to explain a computer phenomenon, it needs to be supported by researched material.

Animal myths and general fallacy can result in creating a scenario that uses something that actually does not exist. All potential models must because of this be supported in sound researched material.

3.1.2 Formalization

Demonstrating the principle is possible through the process of creating models. Approaching a modeling design requires a comprehensive overview over the task ahead. Deciding a modeling language is an important part of this

process, mainly due to the limitations and/or the possibilities of others.

Deciding a modeling language is therefore a key part of the approach. The model must be able to be used as a starting point for Q2 to create an actual implementation of the model. Several different modeling language exists, starting with pseudocode.

Pseudocode is a tool for planning, defining and/or documenting of a program or a routine[2]. It is very similar to real code, but is written and formulated for human interpretation rather than for computers. Due to the technical nature of pseudocode the separation between design and program may be obscured. To understand the model a certain amount of programming expertise must be present.

Focusing to much on the code, and less on the critical components may lead to poorly designed system. Concentrating to much on the code will also limit the creative process of Q1.

Formal notation is a type of different mathematically-based techniques for the specification, development and verification of different software and hardware systems[15]. Formal notation has been criticized for being time consuming, whilst only assuring correctness. Due to the limit timeframe in this thesis and the need for a practical implementation the formal notation is unsuitable.

The need for a graphical presentation allows for better understanding of the underlying processes, allowing to easier see the mechanisms and processes at work. UML or Unified Modeling Language is an example of a graphical modeling language. UML is however been criticized for being ambiguient, inconsistent and incomplete in terms of it's semantics[51]. The usage of petri nets allows for a more formal approach and is more used towards distributed systems.

Petri nets allow for non-sequential processes, and thus allowing the parallelism characteristic of multi-agent systems. However the models created does not necessarily give a good impression on the different components in a system. BRIC or Basic Representation of Interactive Components gives an operative representation of the functioning agent or multi-agent system based on a component approach[28]. The models created with BRIC is very similar to electronic components.

The BRIC models demonstrates the communication flow in the system. This demonstrates how an incident or trigger flows in the system. It shows which action is performed and to which component. The components demonstrates

different scripts, functions or programs that is defined as an entity.

By creating a model it servers as a visual representation on how the biological or behavior can be implemented. It also serves as a guideline or a low-level design specification of the basic functionality of the system. This facilitates the implementation in Q2.

3.2 Q2 - An implementation approach

The implementation or translation of the modeling approach into a solution that can be used is the goal of Q2. The goal is to create a solution that can be utilized by system administrators in known conditions and by existing tools.

Simulation vs real-life implementation

Choosing between simulation and a real-life implementation is the first crossroad in an implementation approach. Simulation methods such as Maude[17] offers validation and proofs. Utilizing simulation allows for the opportunity to create system in a utopian systems without noise or external influences. However the main criticism of other approaches is the use of theoretical models with little to none real-life implementation and usage. This limits their impact in system administration and the model never gets implemented in real scenarios.

Platform - local or cloud

Hardware is a costly factor to consider whilst planning this approach. Buying and setting up the correct amount of hardware to have a realistic environment is time-consuming and expensive. Utilizing the cloud to host our virtual machines have several benefits over a local virtual machine deployment. Using the cloud, the total number of virtual machines can be scaled to an infinite (in theory) number of machines without them competing for the same resources.

In a localized virtual environment the demands to the hardware and the amount of resources would had to be drastically increased to cope with huge spikes. Competing over resources means more virtual machines compete over resources on the same hardware and thus not offering an actual increase in performance. In the cloud the competition is much more regulated and offers less noise in terms of the hogging of resources.

Another reason to utilize a virtual machine or cloud environment is the goal of creating a realistic test-scenario as close to real-life usage as possible. The goal of creating a solution that can be utilized and implemented by other system administrators is easier to achieve through a realistic implementation.

Simulations offers several possibilities, but the lack of realistic interference and noise makes the results unrealistic. A more realistic approach is the usage of real servers providing a real service. An example of this service can be a webservice.

However the cloud does have a set of limitations when it comes to the allowed changes to the virtual machines and its properties. The Amazon EC2 does not support live migration of hosts and does not support the dynamic upgrade of virtual machine hardware on the fly. Additionally, experimenting with scaling and other dynamic behavior would be more costly in a cloud based environment.

Due to these limitations in the cloud, a portion of the initial testing will be executed in a local environment. Deploying and testing a method or function on local hardware allows for quick and flexible changes. When the desired method or function is working, it is ready for deployment in the cloud.

3.2.1 Utilizing existing tools

This type of assignment is very profession-oriented with regards to the desired results. As mentioned previously the need for a solution that can be actually used is one of the key aspects. One method to achieve this is to utilize programs and solutions that are already well known to system administrators. This covers from typical system administrators tools like SSH, but also towards platforms.

Re-using knowledge about tools and platforms does not only lower the threshold of implementation for system administrators, but also minimizes the amount of new technologies to be accustomed to. Utilizing well-known and familiar tools also have the benefit of more confidence and trust. The different software and platforms have existed in larger time-period and the functionality and stability is well known. These different factors increases the chances of the solution to be widely adapted and have a large impact.

3.2.2 Testing

There exists two different approaches when testing in this type of environment, synthetic and real-life traffic.

Synthetic testing

Synthetic traffic would be using computers to generate traffic, as opposed to real-traffic where traffic is generated by users. Using synthetic traffic gives the possibility to execute testing fast and in a repeatable fashion. This provides for more possibilities toward data gathering and proofs. The data gathered would be more easier to predict, and the entire testing could be undertaken in a short time-period. Using synthetic testing the test environment could be simplified to fit the desired scenario.

Real-life traffic

One of the key aspects of this assignment is the real-life implementation of the solution. Because of this, a more realistic test scenario must be designed. Using real-life traffic would have been the most ideal choice towards a realistic testing scenario. However this is implausible for several reasons. Firstly, the amount of traffic to the website would have to be of a certain size in order to be useful in the planned testing. Secondly, the platform and tools used would have to be identical to the tools that were planned in the experiment. Already now the number of possible testing environments have decreased drastically.

The main reason why a real-life implementation would be unsuccessful is the risk of implementing an untried solution on a larger network. Because of this, the two different approaches must be combined. Utilizing synthetic testing methods to recreate real-life traffic creates an optimal test-environment. The data gathered would be regarded as trustworthy due to the use of real-life traffic. Using synthetic testing methods offers stronger predictability and allows replaying certain scenarios.

If the implementation would prove useful and effective the next logical step would be to implement in a production environment. This is however totally dependent on the results from this thesis. A production deployment is unlikely if the solution has not proved to be both functioning, stable and reliable.

Replicating real-life traffic

A more realistic scenario is based on realistic webserver load. Load shifting with several different highpoints and low-points. If a company were to consider a cloud environment, it would be necessary to create scenario as close to real usage as possible. To recreate this, a request was sent to the website "finn.no". Finn.no is a Norwegian marketplace website for both businesses and single individuals. The graph 3.1 was given in response from Finn Labs.

The graph contains no sensitive data, but is not to be used outside this thesis after agreement with finn labs.

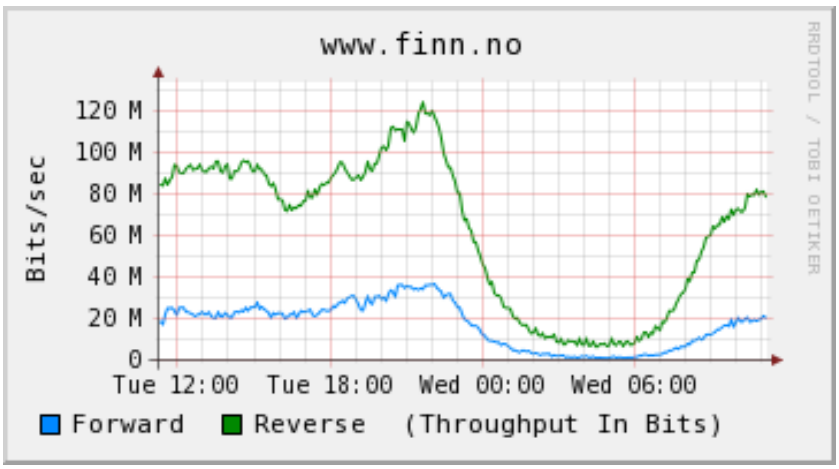


Figure 3.1: An example of daily usage from a the Norwegian website - finn.no [45]

Since no numeric values was supplied only the profile of the forward will be used. The graph spans a 24-hour time period, from 12 in the afternoon till the next day.

3.2.3 Visual representation of tests

Creating a visual representation of the running virtual machines is also a key element that must be created before testing can be started. This visual presentation would have to show the number of virtual machines running at any moment as well as their current status with regards to performance, health and other necessary data. Retrieving information of virtual machines that are powered off, crashed or have died must be possible.

The timeline should be created dynamically and adjusted to changing conditions. This means that it should list the health of a VM, basically it's age and hunger. The timeline should be presented as a graph available either as an report or a webpage.

A log of running machines, startup/shutdown-commands must also be created. This is to show how the different types of scenarios develop, and how the decision-making process is done. The log should use standard time stamps and host information to link actions with hosts.

3.2.4 Expected results

The desired results from Q1 is a number of models that uses either biological mechanisms or animal behavior. The model will explain a new or existing method to manage virtual machines by using biology. Firstly the type of model is introduced, either as a mechanism or a behavior.

Then a parallel to system administration, were it will be compared to a type of technology or platform. If it is a new type of model without a parallel, it will explain how it can be beneficiary to system administration. Then a BRIC model is created to recreate or mimic the desired behavior or mechanism. Finally the BRIC-model should be explained, addressing design choices, benefits and limitations.

Number of models

Creating the correct number of models is important. To few, and one risks not demonstrating the principle well enough. To many could lead to time restraints for the implementation. The models created must demonstrate the principle of all the goals set in Q2. This also include the implementation.

Chapter 4

Results Q1

Creating a set of models that are comprehensive and covers all the fields discussed in the problem statement is a huge challenge. The following biological mechanisms and behaviors was chosen and grouped based on the different focus areas from the problem statement.

- Resources
  - Internal clock
  - Metabolism
  - Circle of life
  - Herd size
- Reliability
  - Herd expulsion
  - Aging
- Security
  - Play death
  - Loosing tail
  - Attack warning

Some of these models cover more than one focus-area, but has been placed where it is most suited. This will be more thoroughly explained at a later stage. The focus-areas is derived from the problem statement Q2. This demonstrates that the two problem statements has some overlap.

4.0.5 Communication

The basis for several types of behavior is communication. This can be communication between animals of the same species. It can also be used as a defense mechanism, mating and many more. Clearly defining different forms of communication that is utilized by animals is important to the different focus-areas.

Animals communicate in several different ways and with different intent. The communication serves multiple purposes, such as alerting other members of the herd, coordinating activities and finding a mate. However there is a difference between signals and communication. Signals can be such as gestures, facial expression, gaze following, vocalization, olfactory communication (smell) and/or electro communication. Communication on the other hand is the collective act of deploying and responding to signals. Communication is a highly debated in biology because communicating is often thought related to intelligence. However the ability to speak does not make you intelligent.

Thomas Sebeok distinguished two fundemental types of signals[55]: Discrete (or digital) and graded (or analogue). Discrete signals are those that operate in an on/off-manner, such as the flashing sequences of fireflies[7]. A defining charachteristics of a discrete signal is that they can only convey a single simple message. They can say that they are aggressive, but not how aggressive they are. Analogue or graded are able to vary intensity and complexity. Examples of this is how ants use chemical responses to alert of danger, and the amount of chemical response is proportional to the danger.

Being able to scale the communication depending on the context is a key element in graded communication. Considering a monkey communicating first through a gaze, but after if it feels more threatened it would open its mouth, bob its head, jump up and down and vocalize among other activities. All this different components requires varying amounts of energy, and is thus costly for the animal. Additionally the animal is at risk of attack from other predators or making the opponent attack premature.

Type of signals

The similarity to the animal kingdom is the need for computer systems to communicate with each other in order to complete tasks, call attention to them or coordinate themselves as a group. This does not encompass software humans use to communicate with each other, such as email and system-to-system protocols. One example is the ARP protocol for ethernet address resolution (a mate responds to a broadcasted call) or messages left in a cen-

tral logging system for others to see (chemical signals).

By looking at system administration several similarities can be found with both characteristics and limitations. Starting with the property range one can think of how different computers communication works locally, in a large network or globally. How well does it scale and how does it perform. The term "Change of message" would be how quickly the technology is able to change it's message from one type of content to another. For example, a queuing system all prior messages needs to be handled before the last message is read. This means that it may go some time before a message is processed by the handler, thus leaving a possible large windows between message sent to handled. This form of communication between systems is asynchronous, allowing the sender to be down or unavailable, but the message remains in the queueing channel. This resembles that of chemical signals.

The ability to go past obstacles can also be similar to that of the ability to pass firewalls or inability to make several jumps through a network. The ability to surpass this obstacles must be an important aspect when choosing a communication method. Locatability in animal life is how easy it is to find the animal that communicated. In terms of communication in computing this would if it is possible to find the sender and the integrity of the message.

Finally is the issue of cost is in animals related to the cost of producing in terms of energy and developing the skill or trait. In computing it would have to cover several aspects. Firstly it would have to cover the cost of sending and receiving the message in terms of computer resources spent. Secondly it would need to cover the cost of implementing a solution, both in terms of skills and potential licensing fees. The cost of maintaing the solution must also be included. Managing SSH-keys in a secure and efficient manner is costly in terms of how much time must be used per machine.

	Type of signal				
Feature of channel	**Chemical**	**Auditory**	**Visual**	**Tactile**	**SQS**
Range	Long	Long	Medium	Short	Long
Rate of changing signal	Slow	Fast	Fast	Fast	Slow
Ability pass obstacles	Good	Good	Poor	Poor	Good
Locatability	Variable	Medium	High	High	High
Energetic cost	Low	High	Low	Low	Low

Table 4.1: Comparing Amazon SQS with biological terms of communication

One example of a service for queue-based communication is the Amazon Simple Queueing Service (SQS). In table 4.1 the Amazon SQS is compared

with the different types of animal communication. The colors demonstrate how the different forms of communication is similar to SQS.

Due to it's long range, low cost and can surpass firewalls it is quite similar to a chemical communication. The potential wait time due to the handling process on the receiving end gives it a potentially low rate of change of signal. Sending signals can be verified with checksums algorithms supplied by Amazon as well as only authenticated hosts can read and write messages.

4.1 Resources

This section will focus on the focus-area of resource management in virtual machine. Resources covers number of servers, load and virtual machines performance.

The internal body-clock of mammals and humans are controlled by a part of the brain called Suprachiasmatic nucleus or simply nuclei[27]. Based on this internal clock, the body makes different decisions. Examples of this is sleep, physical activity, alertness, hormone levels, body temperature, immune function, and digestive activity. This means that the body autonomically changes it's state depending on the time of the day.

Running the human body on 100 % all the time would require large amounts of energy and would require humans to eat large amounts of food. Servers does not have the same limitations as the frail biological bodies. Servers do not need to sleep to maintain readiness, but in a cloud environment the need to preserve resources is a promising prospect. By using the analogy of rising and lowering of body temperature in biology, one can use this knowledge to the dynamic resource allocations in virtual machines. Virtual machine platforms such as Xen and ESX supports the dynamic change of virtual machine hardware, such as changing the CPU or memory. By doing this, virtual machines would know when the traffic is low or high, and automatically change its properties.

Figure 4.1: Pictured - a very ineffective bear [59]

The bear pictured in figure 4.1 is an example of several difficulties in system administration. The bears size relative to the food in the birdcage and the amount of energy used to access the food is un-proportional. This means that the bear would use more resources getting the food than the food will in-fact get it. If the bear was the size of a squirrel, it would use less resources and could survive with less amount of food. Comparing this to a server would be to have a highly over-dimensioned webserver with unused hardware resources.

Traffic, meaning incoming requests from clients would create load on this server. The server would need to be able to handle this load at all times. However the characteristics of a physical machine and the variance of traffic means that a large portion of the time the server is not performing within it's ideal state. If a smaller server had been used it could serve the same amount of users, but it would use less resources which in terms would mean less cost to the business.

Looking at traffic as a food source and the specifications of the virtual machine as its size allows for some interesting prospects. It enables the possiblity of looking at this as an evolutionary adaption. Darwin released "The Origin of Species in 1859"[21] and presented the theory of evolution. In natural selection, animals adapt to their environment through enhancing

or minimizing traits that improve survival and reproduction. Using natural selection on servers allows virtual machines to adapt to varying situations and environments.

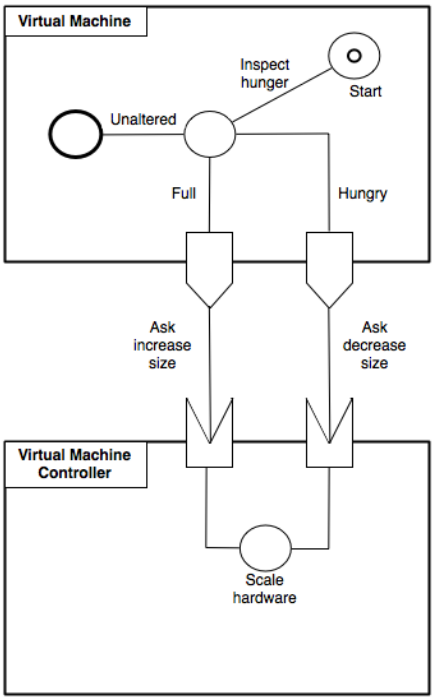


Figure 4.2: Metabolism

By watching figure 4.2 it demonstrates a potential method to change the running characteristics of a virtual machine whilst running. The interesting part of this approach is that the virtual machine itself sends a message to the virtual machine controller to reduce size.

Starvation

Starvation is caused by the unbalance between the intake of energy and used energy. The energy used by the animal is larger than what is obtained through food. Humans that starve over a larger time period can develop organ damage and will eventually lead to death.

4.1.1 Circle of life

Animals as well as humans mature from the time of birth throughout its life. The maturation of sensory, motor and other physical systems as well as

learning are how animals adapt. The constant progress of adaption, learning and maturation is continuous and allows the animal to easily adapt to new situations and environments. By looking at birds and singing it is possible to look at both the physical characteristics of the animal, as well as the adaptations, learning and experiences.

The physical characteristics of the animal are decided through genes, such as size and form. However, experience or learning are also a major part of adaption. Reusing previous experiences to behave in the most beneficiary way gives the animal an advantage over other animals. Learning amongst birds[7] shows that birds that are isolated sings less complex songs than other birds of the same species. By listening to other animals to birds can create more complex song, allowing it to be more attractive towards other females.

The circle of life covers from the birth or creation of the virtual machine to its death. It does not cover the actual coitus, but starts from the moment of conception. The decision to create a new virtual machine is issued by it's parent, i.e. the virtual machine controller. The machines "recipe" or DNA is the settings provided by the VM-controller. This is everything from network interfaces, platform and other technical specifications. Animal behavior is often learned through experience[7]. Adapting to the environment and other behaviors must be learned, and by using configuration management tools the same teaching can be done in system administration.

This presents a nature/nurture-debate of what is most important. Platforms, distributions and physical attributes of a machine are characteristics that are nature (genes). Software, behavior and policies are typical examples of nurture. Network interfaces could be a typical examples of this. The number of, and physical attributes of the interface are based on settings made by the virtual machine controller. This cannot be altered by the machine itself, only by a change in the actual hardware (physical or virtual).

The nurture debate would be the settings of network interface. The details about subnet, netmask or IP, or even if the interface should be running at all. This means that some parts of the genetics (the VM property) that can be heavily influenced or in fact dominate the physical characteristics of the VM.

Aging or senescence is another aspect of the circle of life. Aging can be considered a change of state in an animal after maturity. In biology the theory of aging is still being disputed and several different theories to aging exists. One of the first prominent theories is mutation accumulation[48]. This idea is based on that aging is evolutionary neglect. Animals in the wild live to reproduce and ensure that it's children live on. What happens with the animal after this is evolutionary insignificant since the offspring has

already been born.

However this is just one theory of aging and several other theories exists. By rather looking at what happens with a body when it ages instead of why gives greater insight into aging and how it can be used to manage VMs. Two different theories of how aging happens exists; programmed and stochastic. Programmed implies that aging is based on an internally body-clock throughout it's lifetime.

This would rely on changes in gene expression of systems responsible for maintenance, repair and self defenses. Stochastic aging suggests that the environmental impacts on an animal cause aging. Examples of this could be for example damage to DNA or damage to tissue and cells by oxygen radicals.

Considering the two different types of theories in how animals age, it is similarly difficult finding an ideal implementation in system administration. Introducing aging in virtual machines would present several different interesting aspects. Poorly designed software and other bugs may lead to memory leakage and general un-stability in a machine running over a long time period. Introducing aging would perhaps be a solution to this problem.

Hardcoding into the machines that after a maximum age, it would die is one approach. This approach is very similar with the programmed aging theory in biology. Hardcoding would allow the system to be predictable, with no machines dyeing at the wrong time of the day etc. The shutdown-time could be tuned to be executed at a specific time, thus allowing for more effective usage of for example Amazon instances. This would allow for a more economical approach. However this approach has some vulnerable points, for example the fact that it still need human interference to manually set the maximum age and decide time of day etc.

Hard coding a fixed time, but introducing randomization would allow for a more flexible solution. In a case where all machines are created at the same time, the machines would not die at same time, but rather in the same time period. This would have be statistically determined to find the ideal time period.

A third form of death would be a death by environmentally factors. Examples of an external factor could be diseases or other types of illnesses. This is covered in the focus-area of security.

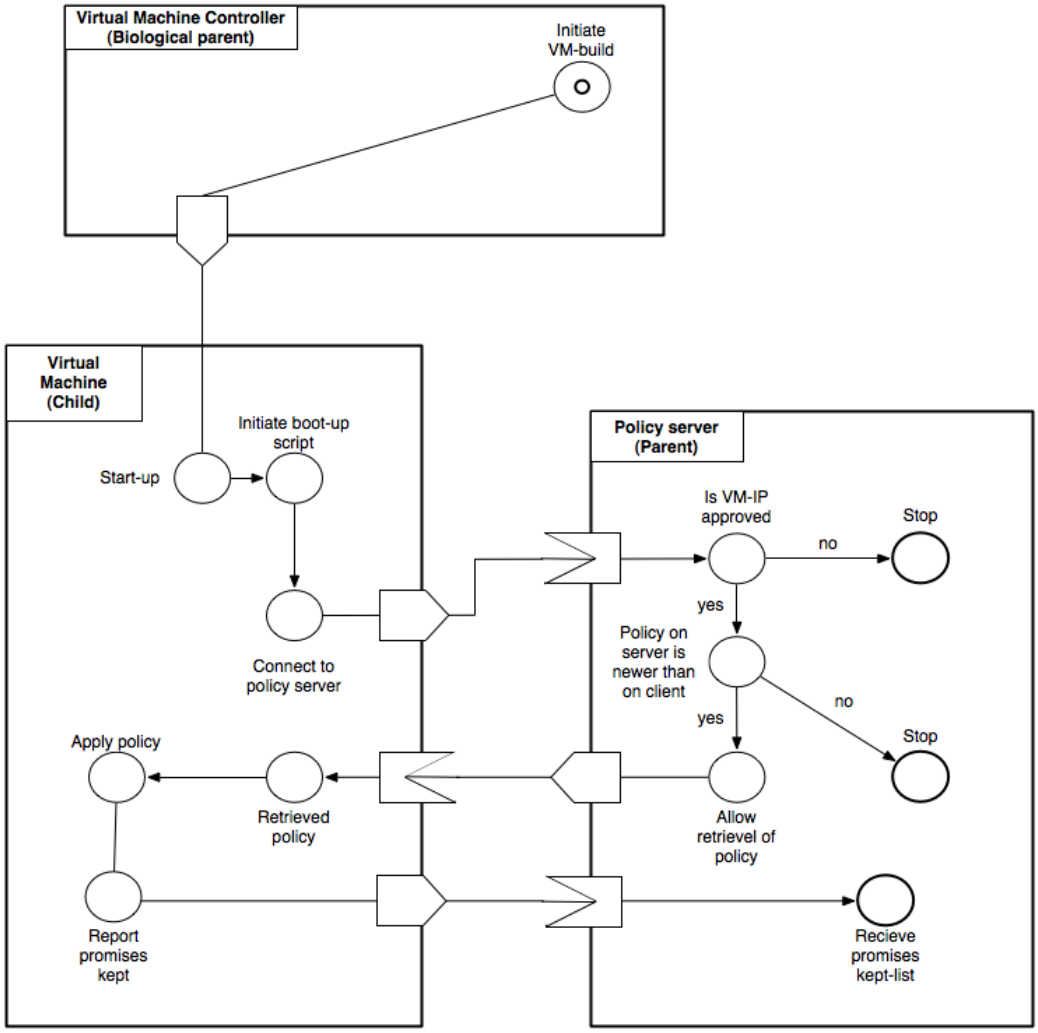


Figure 4.3: BRIC model of Circle of Life

Figure 4.3 is a model on how a machine has conception and adolescence. The virtual machine controller starts the VM with a predefined configuration, represented as DNA. The process of configuring is done through Cfengine, represented as teaching and learning from a parent. This represents the nature/nurture in computing.

4.1.2 Herding

Herds of animals varies in both shape and form. It can be the gazelles on the savannah to the complex structured societies of eusocial insects such as ants. Not all animals live in herds, and there are both benefits and costs of living in a herd. The following are typical examples of benefits and costs:

- Benefits[7]
  - Individual risk of predation diluted by joining a group
  - Grouping confuses predators, making it harder to prey
  - Groups can tackle larger prey collective than individually
- Risks/costs[7]
  - Greater risk of contracting diseases by living in a large social group[38]
  - Greater interference from competitors
  - Greater risk of inbreeding

By looking at this there are several similarities to networks of computers. Whether it is a large network of virtual machines or a load balancing pool it can be regarded as a herd. The simplest example would be that groups can tackle larger prey collectively than individually. Large spikes in traffic is easier to manage for a load balanced multi-server scenario than for a single server.

If a number of servers are connected through the same network the risk of the herd is larger when a single server is infected. If a single server on a system is compromised, the network can be used to spread the disease further. Because of this, animals have developed methods to expel individuals that are sick or have undesired behavior.

4.1.3 Herd size

Living in group has both advantages and disadvantages. However increasing the herd does not necessarily increase the benefit of being in the herd. The competition increases as well as the availability of resources (food). Also the the optimal size of the herd is dependent on the circumstances. Zebras don't react toward single hyaenas since they are prone to hunt gazelles, but reacts to larger groups of hyaenas since groups can hunt zebras.

Finding the optimal herd size is therefore a key factor for the competitiveness of a group. The optimal size is relative to that of the animal and the environment. In Animal Behavior[7] the following graph is presented figure 4.4.

The most advantageous feeding size is reached at an early time and due to competition it slops fast. More animals find food faster, but the competition means that it is not effective enough. The internal struggle for resource and struggle increase with the size of the herd. In essence, the larger herd, more struggle of resources. Scanning refers to the herds ability to scan for predators. This also flattens after the herd increases to much.

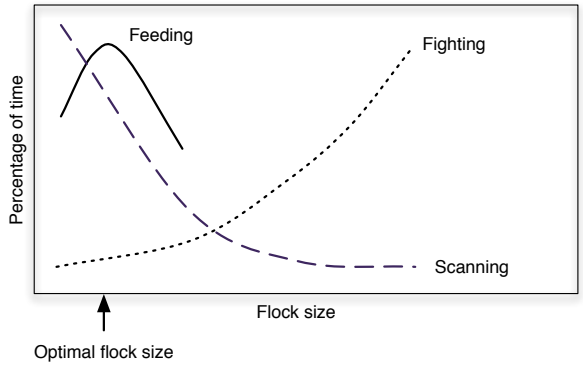


Figure 4.4: Biological presentation of optimal herd size[7]

The graph 4.4 assumes that the animal has two types of risks, starvation (lack of food) and predation (attack from predator). The balance of the herd is then a consideration between the two different risks, however it is more common to have a slightly more animals than there are food available, see figure 4.4

By looking at herds and how animals adapt there are several similarities towards virtual machines and virtual machine management. Looking at food as web traffic and the herded animal as the webservers. Without web traffic the servers would starve and would die of starvation if it remained un-fed.

The internal fight between the servers would be the fight of resources. This would cover both the fight for food (i.e. web-traffic) or place in the herd. Considering a scenario where animals were always fed to their full capacity, the web server would utilize almost it's entire capacity. Only when it has reached this capacity, the next-in-line VM can start eating. If the traffic is low, the least privileged VMs would starve and eventually die.

This is similar to the thought of "survival of the fittest" and would also make sense in an cloud environment. Considering a series of webservers dynamically changing it's size depending on traffic would allow the running

cost to be low. Also there should be a maximum level of the herd. This level should represent both a peak for economic restraints, or as a performance threshold of the technology. Load balancing technology can increase the performance threshold, but it is not infinite.

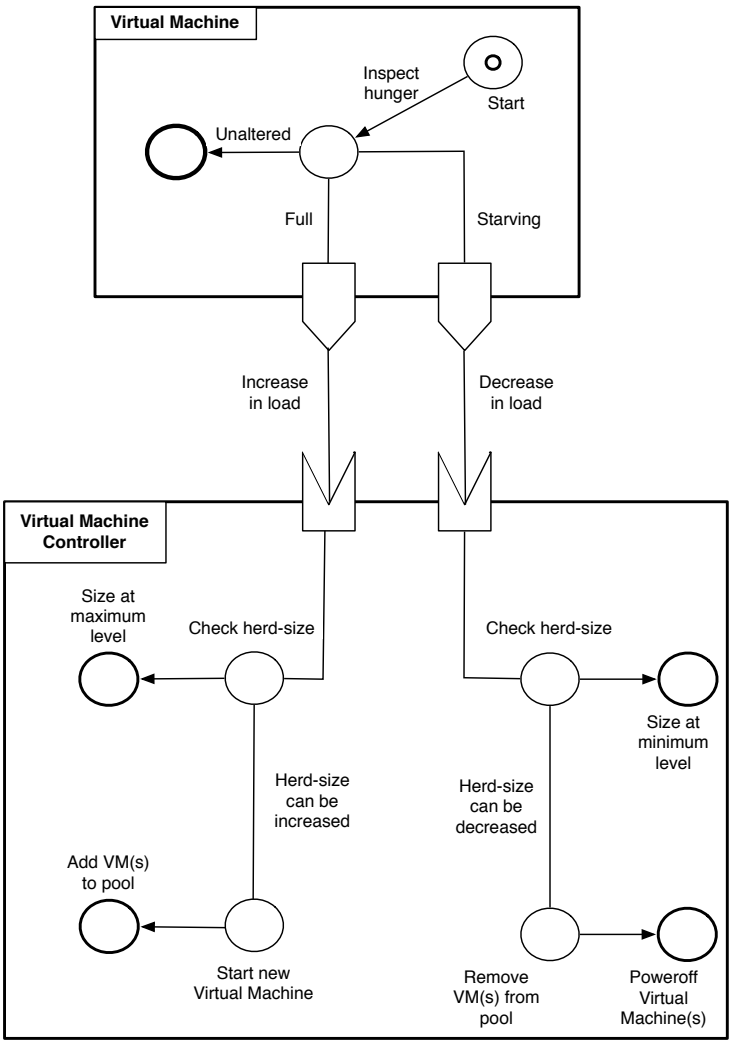


Figure 4.5: BRIC-model of herd size

By looking at figure 4.5 there are several changes done to fit more towards a virtual machine environment. The starvation and death of virtual machines are only up to a point, if the herd has reached is minimum level, it will not die. Thus the species would ensure survival, although it is technically cheating by biological standards. Lowering the resource usage in a starva-

tion scenario would be further demonstrated in metabolism (see Resources - Metabolism).

The same limitation of the herds maximum size also applies. This is a factor that is decided by the system administrator based on technical or/and economic limitations. The virtual machine is what starts the process, thus de-centralizing the decision system. This means that the VMs push their updates to the controller, but it is the decision on the controller whether not a scaling is needed. If there is no change in load, i.e. food, there is no change in "modus operandi".

4.2 Reliability

This section will focus on how to improve reliability in virtual machines. Software and poorly written code can cause a server to reduce performance over longer time periods. This sections seeks to address this issue.

4.2.1 Herd expulsion

Herd animals behave in a group behavior, because if an individual would behave outside the normal behavior would become an outcast. In social hierarchies, like wolf-flocks different animals have different social roles. Three types are most prominent in the wolf herd, the alphas, the subordinate beta and the low-ranking omega[19]. Alphas are not leaders by human standards, but has more individual rate. The omegas have a low-rank, and gets large amount of aggression from the rest of the herd.

Using promise theory, a hierarchy of virtual machines can be created. It is important to note that when speaking of virtual machines in this context they are of the same breed. Comparing webservers with DNS-servers would present a large challenge that is outside the scope of this assignment. But considering a homogeneous group of virtual machines the omegas would be the servers that keep all promises. Machines that do not behave in the way the herd desires, i.e. not keeping the promises would be expelled from the herd. By doing this the undesired behavior is actively removed from the flock, improving system integrity and reliability.

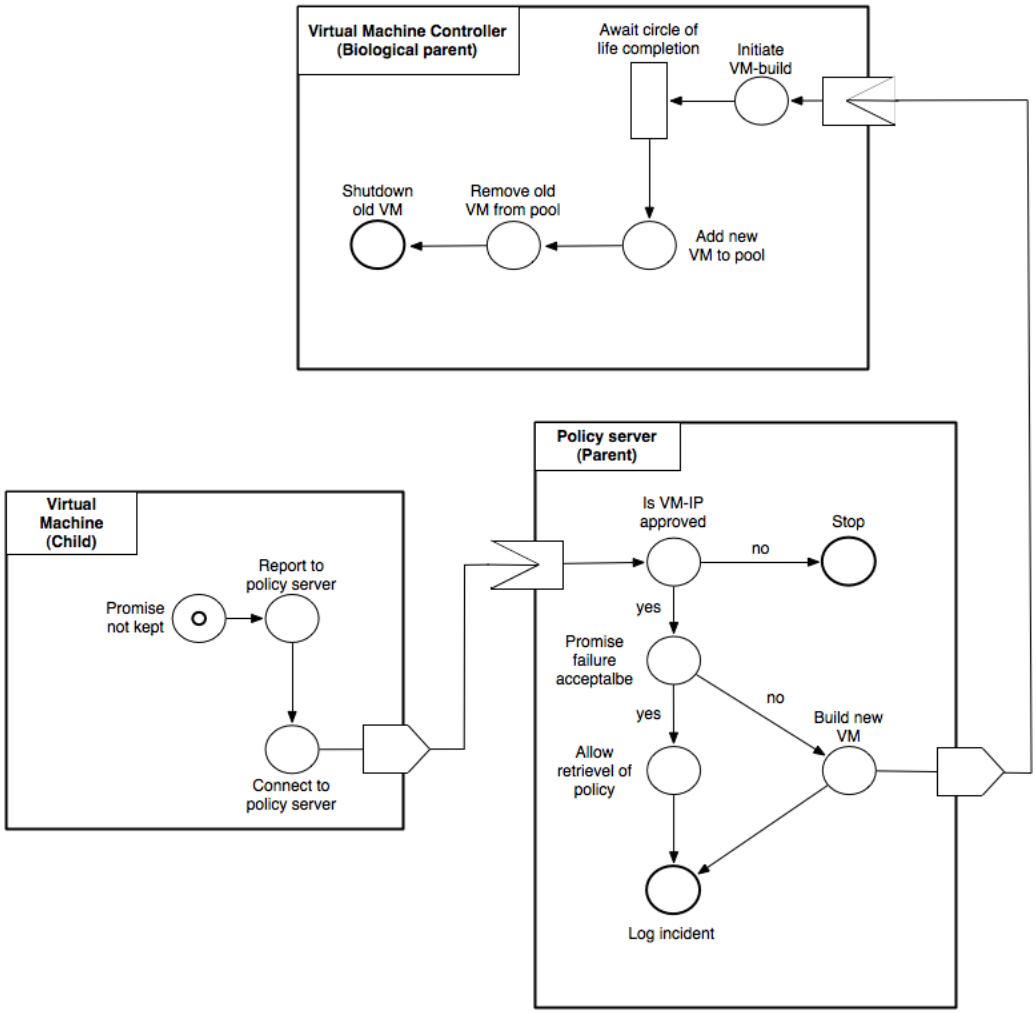


Figure 4.6: BRIC model of Herd Expulsion

Figure 4.6 shows how herd expulsion through the use of Cfengine promises. If a server fails to comply to the agreed behavior it will be expelled from the herd and effectively terminated. In this case the policy-server and virtual machine controller is two separate machines, but this is not necessarily the case.

4.3 Security

In animal anti-predator mechanisms there are two different types of mechanisms[26], primary and secondary mechanisms. Mechanisms and behavior to prevent detection are usually referred to as primary defense mechanisms. Secondary defense mechanisms are the defense measures animals have and use after they have been detected. Examples of secondary defense mechanisms are distastefulness or escaping. Primarily the models created in this thesis will be derived from secondary mechanisms.

Defense mechanisms that relies on the attacked animal retaliating or otherwise harm or injure the attacker is not included. The legality of attacking a predator is disputable, and in a case where normal traffic is detected as an attack it would be highly dangerous. Accidentally attacking individuals or others would be very ineffective.

4.3.1 Feigning death

Thanatosis or the feigning of death is a protective mechanisms to prevent being eaten by a predator. Predators find immobile objects to be uninteresting. Death feigning does not only happen in the animal world, but has real-life implementation such as in computer games (Gears of War). Death-feigning is also a individualistic defense mechanism[61]. The animal does not consider protecting anything but themselves and sacrifices other fleeing animals to the mercy of the predator. This protection mechanism is more common in with animals not living in a herd.

Using this in system administration would be to appear as it is dead. The definition of a dead server would be dropping all network packets and/or the stopping of services. By using a webserver as an example the server would stop all incoming and outgoing traffic through it's network interface. Additionally it would stop the webserver, SSH and other services it may have running.

The largest issue however with a feigning death approach is that of availability. If the server is attacked or considers normal behavior as an attack it would drop all network connections. By doing this, it would not be able to continue serving it's role as a server. The services on the server would be unavailable and Distributed Denial of Service attacks (DDOS) would be very successful against servers where this implementation was implemented.

An alternative anti-predator behavior which bases on a similar approach is the sudden flight behavior in for example grasshoppers. This type of flash and erratic behavior, going from bursts of movement to sudden stop confuses

a predator and can make it loose interest/focus. This could perhaps be a different approach where an attacked server sporadically blocks traffic, thus a minimal amount of service would still be supported.

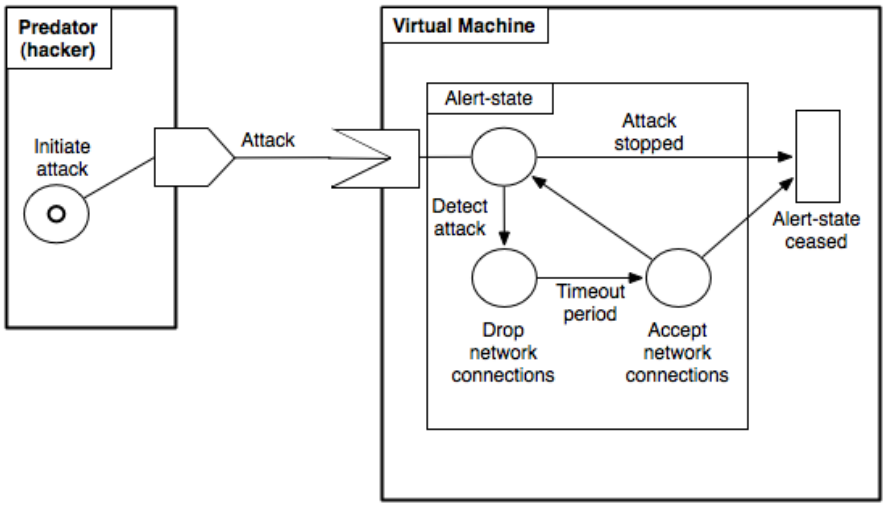


Figure 4.7: Bric model of virtual machine feigning death

Figure 4.7 demonstrates how this implementation could be used to improve security. An attacker initiates the attack on the target system. The attacked machine detects the attack and enters an alert-state where every all types of services are stopped. Only system crucial services are kept running. After a time period the system starts accepting network requests. If the attack is still undergoing the machines stops all services again and restarts the process. If the attacked has seized it exits the alert state and restarts all stopped services.

4.3.2 Loose tail

The power of diversion have been a success formula in animal behavior as well as illusionists performing tricks and magic. Diverting attacks either away from the animal itself or vulnerable areas such as the head gives an animal better chances of surviving an attack.

The loss of tail during an attack is a trait among certain lizards and snakes[62]. This is called autotomy and is an anti-predator behavior. The tail or skin falls of and squirms. The attacker then reacts to the movement of the lost limb and the attacked animal can escape. This approach requires a large amount of resources from the animal, since the animal needs to regrow the lost body part.

To increase security in a virtual machine environment a similar approach can be created. When an attack is discovered, the virtual machine does two actions. Firstly it communicates to the virtual machine controller that it is under attack. The virtual machine then starts the process of replacing the attacked machine. The second act of the attacked VM is to shutdown to a mere minimum of services. This could be shutdown SSH, remove webpages and generally slow the responses to the outside world.

When the new virtual machine it adds itself to the pool and the old attacked VM is killed of. The time between attack and to the time where the new virtual machine is online, the service will be unavailable. However, the attacker might loose interest when the attack has very little effect.

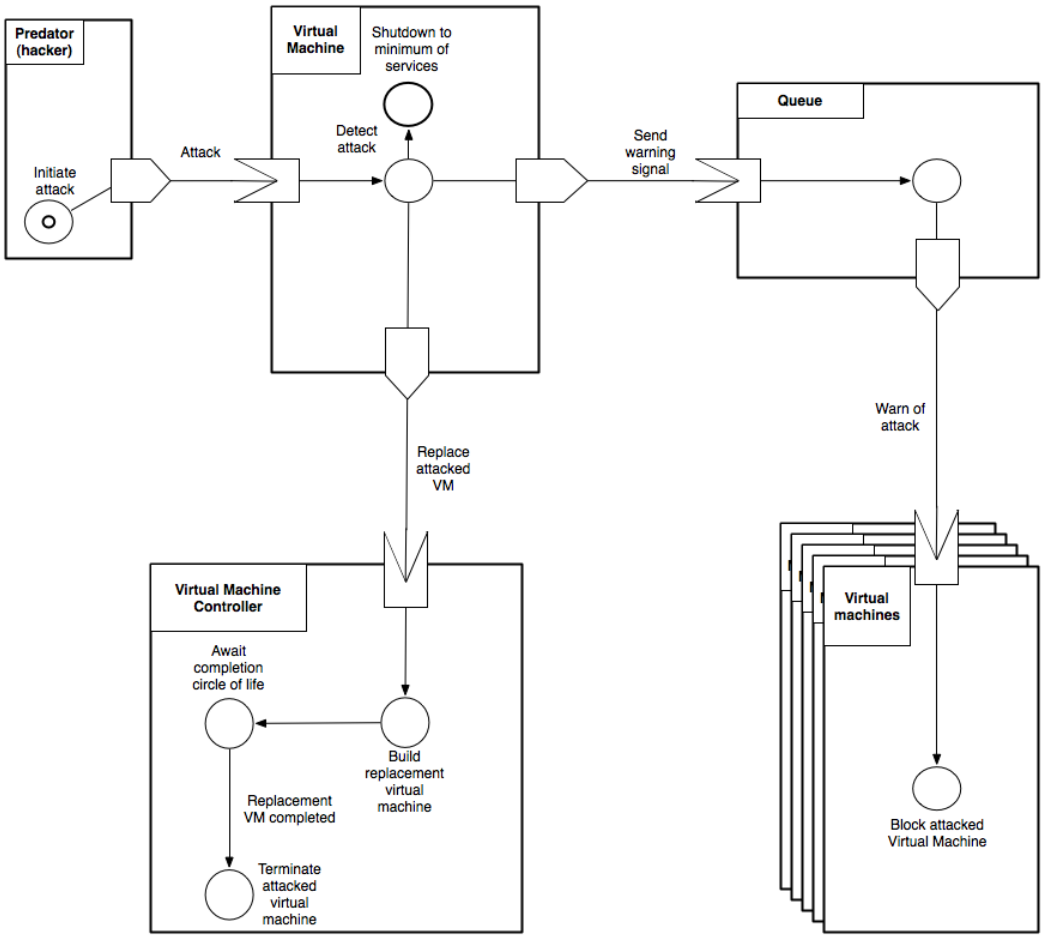


Figure 4.8: Attack warning with loose of tail in virtual machines

Figure 4.8 demonstrates how a machine is attacked and enters an alert-state. This model regards all servers as one joint animal and the attacked server as the tail. The attacked machine sends warning to the other servers and accordingly gets blocked. This separation can be compared to loosing a tail. The body, i.e. the controller regrows (rebuilds) the lost member (server).

4.3.3 Attack warning

Attack warning or alarm signals are another interesting approach for virtual machines to increase security. There are several types of different alarm signals and different effects. Some toads have been known to release chemical alarm signals that both serve as a alarm method, but also works as a deterrent by being noxious[7].

Vocal signals is another type of attack-warning method, but has several key

elements of attack warning benefits and flaws. The flaw is that the sound can be used by predators to attract attention to the herd or individual that made the sound. However the survival of the herd is more important than the survival of the individual. Vocal signals can also serve as a deterrent, when an attacker is discovered the sound may force the attacker to break cover at an earlier time, and thus loosing the element of surprise.

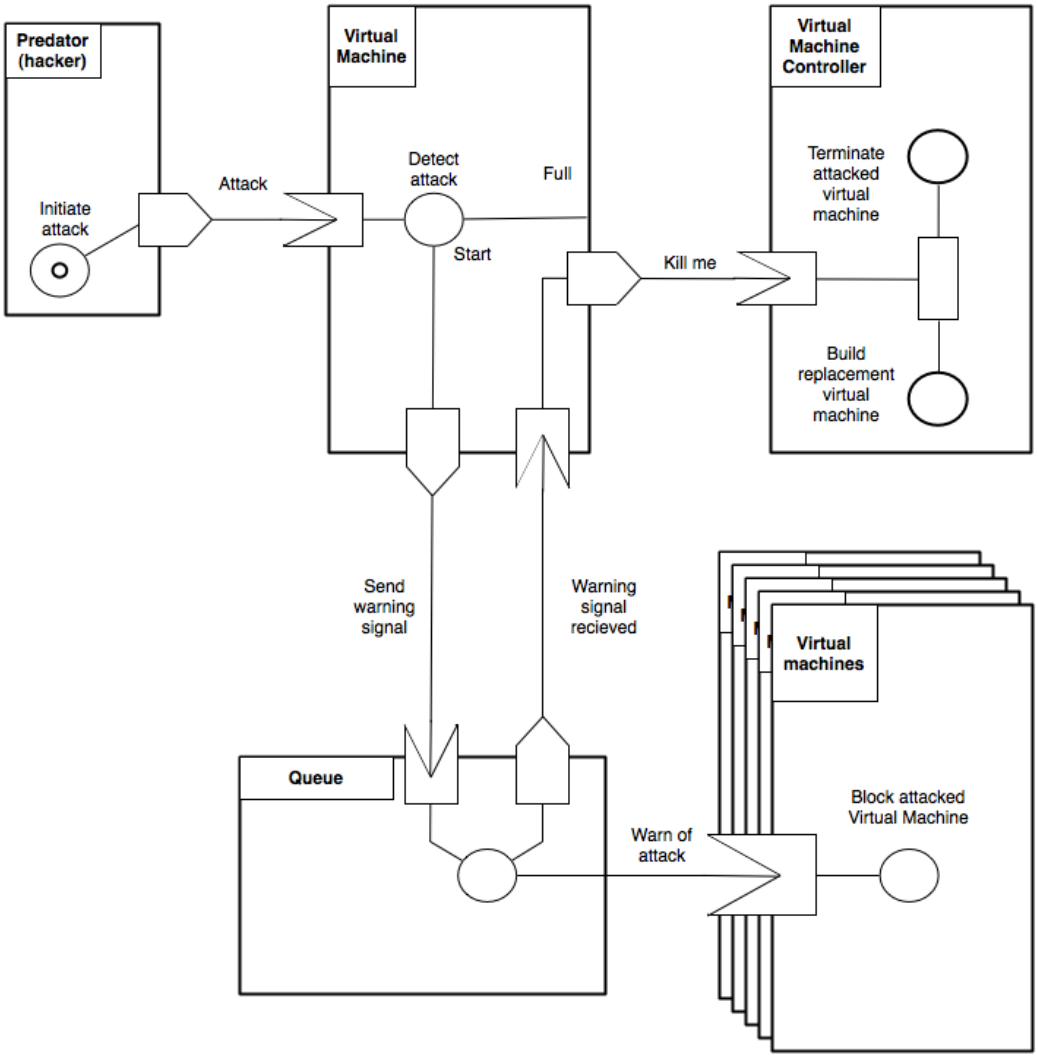


Figure 4.9: Attack warning in virtual machines

The model 4.9 created has a more fatalistic view on the alert system. The attacked VM warns every VM in it's surrounding and when it recognizes that

everyone has got this message it requests to be terminated. By doing this it sacrifices itself for the survival of the herd. The virtual machine controller terminates the attacked VM and builds a replacement.

4.4 Models to implement in Q2

Which models to implement is chosen based on the following guidelines.

- Ease of implementation (plausibility)
- Coverage in Q2 (Resource, security or reliability)
- Realistic (could be used in a real scenario)

Using these guidelines the following models were chosen for implementation:

- Herd size
- Attack warning
- Aging

	Focus area		
Concept	**Resources**	**Security**	**Reliability**
Attack warning	no	yes	no
Herd size	yes	no	yes
Aging	no	no	yes
Overall	**yes**	**yes**	**yes**

Table 4.2: Comparison between focus-areas and models to implement

Figure 4.2 is the selection of the models to implement in Q2. No model covers everything, however all models combined covers all different focus-areas.

Chapter 5

Results Q2

This chapter covers the results chapter of Q2. It is the actual implementation of the selected models from Q1. It spans from planning, development to testing of the different solutions.

5.1 Platform Choices

Due to the price of creating a large virtualized environment, all testing should be done in the Amazon cloud. However a large portion of the initial testing and deployment was done in a localized setup to save time and resources.

It was important to reduce the amount of time between a testing scenario had been completed locally, till it could be tested in the cloud. The tool MLN makes it possible to change between a localized setup (xen, vmware or UML) or a Amazon-setup with only a few commands. This made MLN the main tool to deploy the virtual machines.

5.1.1 Local setup

The local setup is based on the Xen hypervisor. In the Amazon cloud, each machine has its own public IP-address. Due to IP-limitations in the local network, a NATed setup had to be used when deploying a large network. This is however the only difference between the topology between a local and a cloud-based setup.

5.1.2 Cloud setup

The architecture and design of the Amazon cloud makes it necessary to add some key features to have a successful setup.

Storage in the Cloud

In the case of a crash or shutdown of an Amazon virtual machines instance the changes to the image is not saved. This means that the changes done to the instance could potentially be wiped at any given moment. This would perhaps present itself as a problem in many cases, but it is rather a case of changing the standardized setup to work better in a cloud environment. Installation and configuration of files either has to be created at the time of the image creation or it has to be configured at a later time.

By using Cfengine to make sure that a new system is configured and behaving in a desired manner is one method of ensuring the state of the VM. Storage of files such as user data can be stored in the Amazon EBS or Elastic Block Storage. This storage is permanent and is not deleted at a crash or shutdown. Continuing the process of crashes or unforeseen issues in the integrity of the VM a closer inspection of the log-files may be necessary.

Storage of log-files

Due to the design of the VMs in the EC2 cloud log files will be deleted in a potential crash or shutdown. Saving the log-files at an external location is thus not only a security aspect, but an absolute necessity. Log-files are important in debugging and an vital part in a development process. Creating a framework for the storage of virtual machines log-files is essential before testing or implementation can be started.

5.1.3 Amazon SQS

The basis for communication between hosts and between host and the main server was done through Amazon SQS. Amazon SQS supports naming of different queues. These different types of queue will be used as channels where different types of messages is sent.

Examples of different queues will be status and system. The status-queue will be used to communicate status messages, whilst the system-queue will be used to send typical log messages. This ensures that the correct message is read by the correct entity.

5.1.4 Scaling virtual machines

An important question is why is it desirable to scale virtual machines. Having all machines running at all times would make sense by just looking at the

performance perspective. However rising power prices, as well as the running costs leads to the desire of having as few servers running as possible.

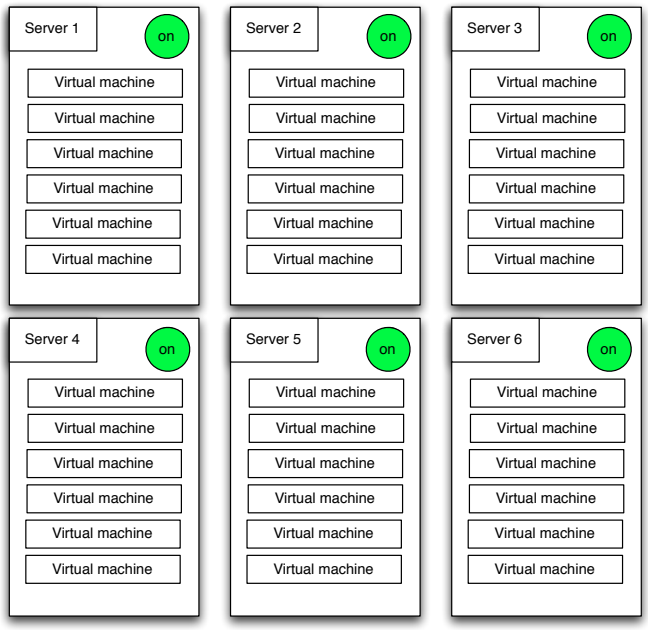


Figure 5.1: High load on data center

Considering the following scenario in a larger datacenter. A total of eight servers running several machines are running, see figure 5.1. They are now performing at the most beneficial state, since all servers are running at the max load.

In times where the traffic to the server is lower, typically at night time or in weekends/holidays, the load on the system is lower. If the amount of servers could be decreased, the number of physical machines can be reduced.

By using migration the remaining servers can be pooled together onto a selected few physical machines. The remaining servers can be powered off, until the load rises. This leads to more effective use of the virtual machines and less resources used to power unused physical machines.

However in Xen the virtual machines can never have more memory together than the physical host. This means that even if the load is very low, the VM has a reserved amount of resources at its disposal. By lowering the reserved amount of resources, more virtual machines can be stacked onto a single piece of hardware. Whenever the load increases, the servers specifications can be increased.

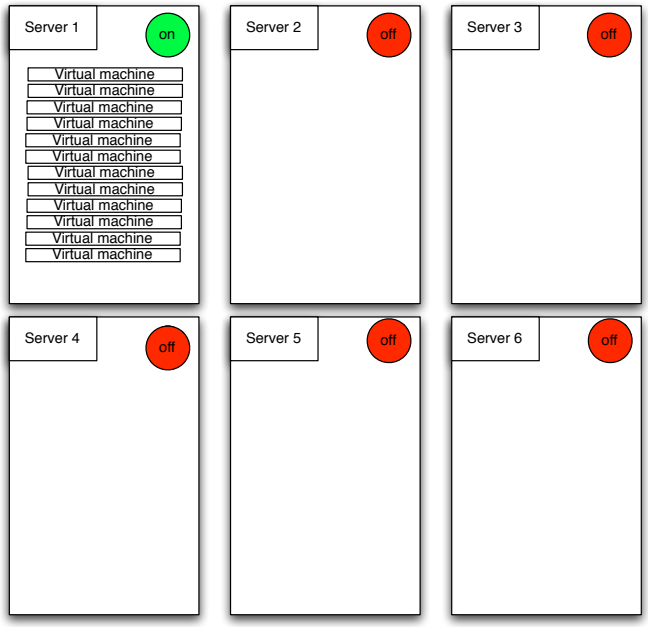


Figure 5.2: Low load on data center

By watching figure 5.2 it is possible to see how the number of virtual machines are reduced. The machines shrink, meaning they reduce the amount of reserved resources, thus allowing more servers on a single piece of hardware.

A death of a VM

Whenever a machine shutdowns, a list of specific commands are executed. This could be umounting file systems, removing the machine from a working loadbalacing pool or similar. Getting the information that a machine is dyeing or have in fact died is vital for the managing virtual machine host. The information should be sent through secure channels to the host.

A virtual machine could also be killed twice. Though this is a logical breech, the need to ensure a machine is dead is important. Firstly the machine should decide to die when its time has come. However the machine may or may not poweroff as intended due to problems in shutdown, a security breach or similar. Due to this the virtual machine controller should kill the VM after a certain graceful shutdown time-period has expired. If the machine is compromised by an attacker, it should not wait for the grateful

shutdown period to be completed.

5.2 Defining starvation

Choosing a webserver

Several different types of webservers exists, such as the Apache HTTP-server[30], Microsoft IIS, nginx and lighttpd. Maintaining a focus on using tools that are actively used in the system administration field, the Apache and Microsoft stand out as the most popular with a combined market share of over 80 percent[41]. The Microsoft IIS webserver is a proprietary solution limited to the Microsoft platform, it has also a 20 percent market share.

In comparison the Apache web server has a market share of over 60 percent[41], is not proprietary and is cross-platform[30]. The Apache webserver has support for languages such as Perl, Python, Tcl, and PHP and extended log-functionality. The Apache Webserver is deemed the most suited platform for this thesis due to it's popularity, cross-platform availability and functionality.

5.2.1 Benchmarking a webserver

Benchmarking webservers can be done in several ways. Httperf gives the possibility to benchmark a system with a number of connections and requests in an easily configured manner. The goal is to find when a webserver starts performing poorly, which can be several different elements such as timeouts, high latency or errors. The monitoring of errors is by far the most obvious solution to find a saturation point. Monitoring errors and time-outs gives the most accurate benchmark scenario.

The Apache webserver also have a module for printing out a status report of the webserver. The module is called mod_status, and is included in the standard installation of Apache[31]. This report prints out either a machine readable file, or as a webpage. The status includes (excerpt):

- The number of worker serving requests
- The number of idle workers
- Averages giving the number of requests per second, the number of bytes served per second and the average number of bytes per request

The number of idle workers is dynamic. This means that the number of workers will rise and fall depending on the load. If the number of idle workers were too high, the webserver would reserve and use more resources than

needed. If the number of idle workers is too low the webserver would not be able to cope with spikes in traffic.

5.2.2 Reporting server status

Reporting the status of a virtual machine can be everything from a few datapoints to large and complex data schemes covering everything from network and disk performance to the characteristics of each program. Due to the limited time and secondary importance of this element makes the lesser and easier approach most attractive.

Amazon offers a solution to autoscale their own services, but this does only work in the Amazon cloud. If a localized setup is used, it would be impossible to use Amazon autoscale or Amazon cloudwatch. The need to develop a tool that works on all platforms is necessary.

Generating load on a webserver depends on the service the webserver is providing. Typically a file sharing webserver would generate more load on the disk, and less on CPU. A dynamic webpage requiring processing on the server side would generate more load on CPU and perhaps memory, but less on disk. The dynamic web page represents more the type of traffic that is the goal of this thesis.

By getting information from the commands vmstat and free. By listing the percentage of CPU being either in user time, kernel, idle or waiting for I/O it is possible to get accurate feedback on the performance of the virtual machines CPU. By using the free-command it is possible to calculate the used percentage of memory.

However the usage of vmstat and free requires varying amounts of scripting to collect the desired data. If more data is to be collected the data collecting script needs to be changed drastically. It therefore was necessary to find a perl module that can collect a vast amount of data with as little change to the original script as possible.

By using the Sys::Statistics::Linux[54] available freely from CPAN it is possible to collect data from a Linux-host. The data spans from information on the system, memory, cpu or disk performance. This allows the script to send status-updates based on the type of webserver.

```
1 #### Only required line to collect user cpu-time
2
3 my $us = $cpu->{'user'};
```

Generating load on webserver

By using PHP it is possible to generate dynamic webpages. A simple php-script reads a file containing a collection of randomized charachters on the webserver and sorts all characters to acceding order. This produces load on the webserver. Depending on the webservers specifications, the load can either be increased through the increase in requests per second, or by increasing the number of characters and thereby increasing the files size of the read file. This would increase the number of characters needed to be read, thus increasing CPU and memory load.

Reporting webserver status

Amazon has a tool called cloudwatch which offers metrics on server performance. The free version of the service gives information at five minute intervals. If paying a subscription, this is increased to one minute cycles.

Getting information of the performance of the actual webserver is not possible by using the Sys::Statistics and thus needs to be collected using a different method. By writing a simple perl-script it is possible to extract the data from the Apache mod_status. This does however have some limitations; firstly that the Apache webserver has to be running to get data. This means that if a webserver crashes, it is not possible to retrieve any data. Secondly, under high load the time between request and answer will increase. This means that under high load, the data may be inaccurate with up to a number of seconds.

By using time-out values it is possible to assumes that if a webserver is not responding, the number of idle workers is 0. Although this resolves the issue regarding load, it means that the messaging system will not always report on schedule due to the difference in load and response time. In a case were a server is running, but the webserver is down, this will inform the system. However it would not be possible to differentiate if a server does not respond due to heavy load, or if Apache is crashed.

The server status is retrieved by collecting the file "server-status" from the webserver. By using wget and timeouts it is possible to reduce the time between request and timeout. This timeout value would be similar to that in httperf, where timeout values are set when the server uses to long to respond.

5.2.3 Improving script reliability

The script created was originally intended to be simple perl-script that was initialized by using cron. The amount of updates could be set with sleep-values and how often cron should run. Under testing with large amounts of traffic it became apparent that this approach had some serious limitations. Amazon SQS requires the initialization of the queue before the message is started. This contacts the Amazon servers and gives the script a "green-light" to send messages. In the first approach this was set at each time the script was started, which would be every minute.

The challenge of this approach was that when large amounts of traffic on the webserver choked the performance, the script was unable to connect to the Amazon servers. This meant that if the server was under heavy load not only the clients received errors, but the server was unable to communicate this to the proper channels.

By daemonizing the script it was possible to set the communication channel at an early stage. The webserver needs to have successfully communicated to the main host that it is up through SQS before it can start receiving web-requests. The daemon also checks if another daemon-instance of itself is running at startup, thus eliminating the possibility of more than one daemon running at the same time.

5.3 Receiving agent

The messages sent from the range of virtual machines is received by a collecting agent on a central server. When reading status messages it is data with two different aspects. Firstly the data has a very low expiration date. From the time the data is sent to it is received and processed must be as low as possible to be accurate.

Secondly the raw data is not necessary to be read again. The mean or the median result of a dataset is of interest to the decision making system. Because of this factors, the receiving agent will delete the messages once it has been processed, thus not allowing the data to clog the messaging system. Amazon SQS does not read messages in the order it was sent, so clearing the number of messages is crucial.

Although Amazon SQS supports large amounts of data to be transmitted per message, the messages sent in this experiment is very few. It is a comma separated list with a hostname, a timestamp and a collection of the performance figures.

```
1 # Typical example of message
2
3 #hostname,timestamp,user,nice,system,id,wait and total_cpu,↩
      used_memory_percentage,free_apache workers
4 host1_shelter,1302357876,5,0,1,79,13,20,95,6
```

5.3.1 Calculating average load

After reading a number of messages the collecting agent can produce an average or median result on each dataset for each host. The script does not allow the data to be analyzed before a minimum of dataset for that specific host is collected. By doing this, temporary spikes in load will not have a drastic effect on the analyzed results.

Through the use of triggers it was possible to guarantee that a minimum amount of datasets were present before the analysis started. If a machine did not have enough datasets it would not give any mean. This means that the calculating agent must have an in-built function to handle this behavior.

The number of messages read is a crucial part of the design and the decision-making progress. If the number of messages read is low, anomalies in load can change the average. This would again mean that the scaling-decision would be done at a much more aggressive rate. Reading a larger set of messages reduces the chance of single anomaly drastically change the average. The drawback of this solution is that it is slower to react to changes in the load.

The advantage of reading a short amount of messages and thus having an aggressive scaling approach is that it can easier handle traffic that varies quickly. The disadvantage of this approach is that it risks starting up virtual machines unnecessary which again adds to the running-cost of the solution.

The advantage of reading large amount of messages and thus having a more abstemious and restrained approach is that it does not scale as aggressively. This again saves money and resources. The biggest drawback with a restrained approach is that it does not cope well with sudden spikes of traffic and by doing this, it risks timeouts for requesting clients.

Which is the best solution of the different approaches is more complex than a simple number. It depends greatly on the current infrastructure, types of traffic and what type of service that is provided. It is therefore much more a business- and policy decision than a programming decision. The script created allows the user to easily change the number of messages.

The ideal solution would be to combine these two different approaches as much as possible. This was made possible through the use of two values, one short and one long. The short interval is used by the calculating agent to increase the number of machines. If there are sudden spikes in traffic, it would be able to respond quickly to the change in workload.

The long interval is used to calculate the number of machines to decrease. Since one do not want the machine to be powered off as aggressively as it is powered on, the window must be larger. This means that the servers have to have a long time period of low load before the machine starts decreasing the amount.

The averaged values were saved to file by using the hostname_short/long in /tmp. The values stored is the hostname, timestamp of the calculation and the average for each value. Finally the script was daemonized to simplify the testing process.

5.3.2 Sliding window

Implementing a sliding window approach allows the analysis of the data to be done continuously. The alternative would be a periodical approach where the collecting agent would collect a set of data during a time period and then calculating the average. This approach would perhaps be less resource demanding on the client, but would be slow to react to changes in load.

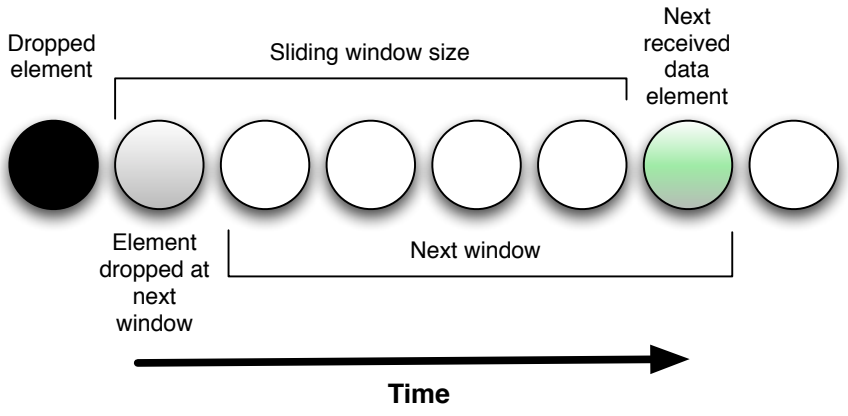


Figure 5.3: Sliding window

By using a simple for-loop through an array it is possible to create a sliding

window with perl. A trigger is activated when the maximum number of elements have been reached and resets the write-to position to 0. The position 0 is then the oldest element in the array, but the newest element then takes it place.

By looking at figure 5.3 the principle of sliding window is demonstrated. Due to the use of arrays it is however not an ideal representation of the implementation used. Because the array is looped, the figure is actually a circle, continuously circling and updating. The data collecting does not however loop.

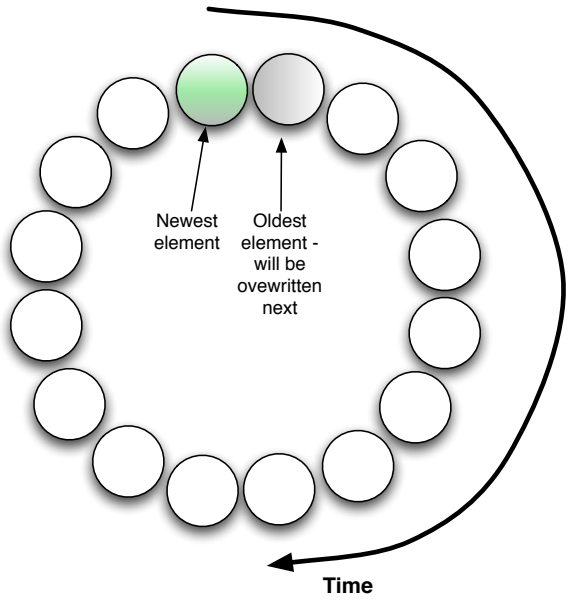


Figure 5.4: Ring buffer

5.3.3 Data consistency

A challenge that appeared with this approach was that the script did not know whether a server was down or up. This meant that a server could be up for a period of time and were able to calculate a mean. If the server was then powered off the script had no way of deleting the old data or resetting them when a new server came up. The old values from the last iteration of the server was present, causing a potential fault in calculating the correct mean.

This problem was solved through the use of SQS poweroff-messages. At shutdown the computer would send a message of shutdown to the "status"-queue. When this message was processed at the collecting agent, it would

delete all values connected with this host, reset all triggers and delete the loadfiles in /tmp.

5.4 Scaling agent

The scaling agents main task is to scale the number of webservers up or down based on the webserver load. By calculating the mean of all the running hosts it is possible to get an accurate description of the load. It would be possible to use maximum values instead, but this would give a much more aggressive scaling than desired. In our scenario the data is homogeneous, thus the difference between the servers will be minimal. The goal is to see if the concept would work, so using large amounts of time to have the most accurate scaling decision is not deemed necessarily.

In cases where a host has just been started the short and long interval may not have been reached. This means that for a period of time, it is not possible to know the state of a given machine. If the load value was set to 0, the agent can in a worst case scenario scale down without knowing the actual load. To solve this a default value is provided that is larger than the value to scale down. The short interval is less likely to be affected since it is reached earlier and by given it the same value one prevents massive scaling.

The scaling agent must know the following information about all machines at any given time.

- Machine status (on/off)
- Machine characteristics (name,IP)
- Machine state (booting/powering off)
- Machine load

It is important to know if a machine is down or up because it shows how many machines that are available. By using MLN it is possible to extract information about hosts. If running the Amazon cloud it is possible to get both hostname and IP-address, but this is not possible with other platforms such as Xen. Also it is impossible to know the state of the machine. A machine can appear online, but can be either booting or shutting down. There is a specific need for a solution that is able to collect these states.

By adding a simple perl-script in the rc.0 and rc.local it is possible to have Amazon SQS send a message whenever it boots or power offs. By simply setting a machine in a state when a command is given (on/off) and first changing its state when a message confirms provides a good solution for

monitoring state.

Using the different run-levels it is possible to predict several aspects of the behavior. The shutdown-script is executed before the other shutdown commands. At boot the rc.local is run at last after every other startup-command is completed. This makes sure that the webserver is actually started before it is declared ready by the scaling agent.

Based on the captured IP-address the scaling agent can add or remove the IP-address from the loadbalancers pool. By using Perlbal one can dynamically change the pool without restarting the service. Perlbal uses telnet to allow this change. Due to the security concerns of telnet, SSH-tunneling was utilized. By connecting through the loopback device it was possible to improve security without compromising functionality.

5.4.1 Rotation

Since servers can perform poorly over time there is a need for creating a rotation mechanism. In this scenario this is done quite aggressive with removing a machine every twentieth iteration. This is mainly due to show the functionality of the solution, but it would be easy to change the script to a larger cycle

By storing a timestamp of when a machine is online it is a simple task of looping through all running hosts to find the oldest living specimen. If the machine is the last of the herd it will not shutdown to ensure the survival of the herd. The rotation does not include machines that are booting. This means that even if 10 servers is being started and only one is currently running properly this will not be stopped.

5.4.2 The scaling decision

The agent now knows how many servers that are running or are off. It know which machines that is currently being booted or is powered off. The machines are added and removed from the pool depending on the state of the machine. The next goal is to create a scaling mechanism that can scale up-/down depending on how weighted the load is.

At the same interval as the rotation of the machines a scaling decision is made. The intervals can be changed, but to shorten the test-time the interval is 10/10 with a sleep time of 10. This means that every 100 second a scaling decision is made. Since it is only two intervals it goes 200 seconds between every scale-up decision meaning quite rapid response-time.

The ferocity of the scaling is dependent on the load. Based on the current load it can scale 10/25/50 percent increase of the currently running hosts. Since it is at this given time impossible to start 1,5 of a machine it must round up to closest number. Demonstrated in figure 4.7 shows how the scaling agent will scale depending on load and current number of running machines.

	Number of machines to boot		
Running machines	**10%**	**25%**	**50%**
1 machine	1	1	2
5 machines	1	2	3
10 machines	1	3	5

Table 5.1: Different scale count dependent on running hosts

This shows that it scales quite drastically different at large number of servers compared to a lower amount. This approach does not cover the potential bottleneck of the loadbalancer.

5.4.3 Scaling down

Scaling down is an important factor of the scaling script. Due to the rotation a machine will always be terminated. But in the case where the long load has been reported to be lower than a given minimum threshold it will decrease by 25 %. This means that if a large number of servers are up, and there are no traffic, it will go gradually scale down towards the minimum amount of servers.

5.5 Status tool

Developing a tool to visually present the load, scaling and the traffic on the servers was necessary. Using MLN it was possible to know the status of any host at any given time. By collecting the data gathered from get_status.pl it was possible to have an accurate reading of the servers load. By connecting to the load_balancer it was possible to get an estimate on the number of requests per second.

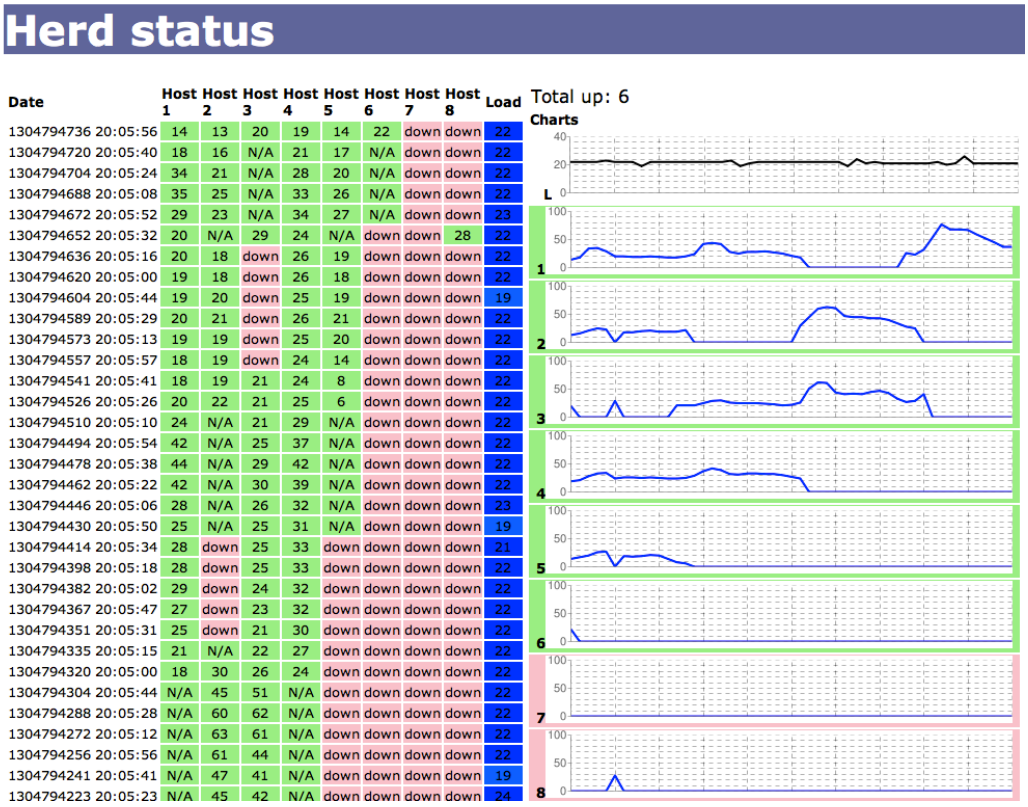
Using PHP[39] it is possible to create dynamic webpages to represent changing data and graphs. By using PHP and Perl together it was possible to create a dynamic view on the number of webservers and load. The current

status is printed out with the newest information first, serving as a monitoring as well as history tool.

The tables are used to present the data in an orderly fashion. The use of colors to indicate whether a host is down or up. This type of color code alert system can improve the comprehensibility of an information system[37]. By simply using colors and text, the total running hosts and the history is more comprehendible than just plain text. The initial goal was to have more information about each running host in terms of load and age. If the average load has not been calculated, the tool simply outputs N/A (not available).

This was however disregarded due to the added complexity of this task and the contradiction of having a to detailed tool. The initial task was to reduce overall complexity of virtual machine management. Having a more complex monitoring tool is practical in a decision making process, but the goal is having the machine manage themselves without human interaction.

Using Google chart[35] it was possible to create charts to illustrate the load on a server. The use of tables and colors around the charts further demonstrates the principle of color-coding the servers status. Online servers are surrounded by green, using color to catch the attention of the user[37].



Herd status

Total up: 6

Charts

Date	Host 1	Host 2	Host 3	Host 4	Host 5	Host 6	Host 7	Host 8	Load
1304794736 20:05:56	14	13	20	19	14	22	down	down	22
1304794720 20:05:40	18	16	N/A	21	17	N/A	down	down	22
1304794704 20:05:24	34	21	N/A	28	20	N/A	down	down	22
1304794688 20:05:08	35	25	N/A	33	26	N/A	down	down	22
1304794672 20:05:52	29	23	N/A	34	27	N/A	down	down	23
1304794652 20:05:32	20	N/A	29	24	N/A	down	down	28	22
1304794636 20:05:16	20	18	down	26	19	down	down	down	22
1304794620 20:05:00	19	18	down	26	18	down	down	down	22
1304794604 20:05:44	19	20	down	25	19	down	down	down	19
1304794589 20:05:29	20	21	down	26	21	down	down	down	22
1304794573 20:05:13	19	19	down	25	20	down	down	down	22
1304794557 20:05:57	18	19	down	24	14	down	down	down	22
1304794541 20:05:41	18	19	21	24	8	down	down	down	22
1304794526 20:05:26	20	22	21	25	6	down	down	down	22
1304794510 20:05:10	24	N/A	21	29	N/A	down	down	down	22
1304794494 20:05:54	42	N/A	25	37	N/A	down	down	down	22
1304794478 20:05:38	44	N/A	29	42	N/A	down	down	down	22
1304794462 20:05:22	42	N/A	30	39	N/A	down	down	down	22
1304794446 20:05:06	28	N/A	26	32	N/A	down	down	down	23
1304794430 20:05:50	25	N/A	25	31	N/A	down	down	down	19
1304794414 20:05:34	28	down	25	33	down	down	down	down	21
1304794398 20:05:18	28	down	25	33	down	down	down	down	22
1304794382 20:05:02	29	down	24	32	down	down	down	down	22
1304794367 20:05:47	27	down	23	32	down	down	down	down	22
1304794351 20:05:31	25	down	21	30	down	down	down	down	22
1304794335 20:05:15	21	N/A	22	27	down	down	down	down	22
1304794320 20:05:00	18	30	26	24	down	down	down	down	22
1304794304 20:05:44	N/A	45	51	N/A	down	down	down	down	22
1304794288 20:05:28	N/A	60	62	N/A	down	down	down	down	22
1304794272 20:05:12	N/A	63	61	N/A	down	down	down	down	22
1304794256 20:05:56	N/A	61	44	N/A	down	down	down	down	22
1304794241 20:05:41	N/A	47	41	N/A	down	down	down	down	19
1304794223 20:05:23	N/A	45	42	N/A	down	down	down	down	24

Figure 5.5: Screen capture of status-tool webpage

Representing the number of requests into the system is also a key element. In this experiment the simulated traffic is quite uniform, meaning that requests are identical to each other. This means a 50 percent increase in requests should give a 50 percent increase in load. This means that the current and former load is of great interest. Figure 5.5 shows what the status-tool looks like.

5.6 Age

Running the scaling script without generating any load demonstrates the principle of aging or generations. By looking at figure 5.6 it is possible to see how the rotation mechanism and the different states work. It originally starts with one machine running. At the first iteration a scaling decision is made, and a machine in the cloud is booting. The time period called "Adolescence" is the time beween the startup command was given to the machine is added to the pool.

The next interval represents the rotation of the oldest living server. Since

only two servers are running, only one machine is powered off. The time between shutdown-command and actual shutdown is called senescence, and represents the time the server uses without a purpose. The machine is removed from the pool at the same time as the shutdown-command to ensure the pool is as up to date as possible.

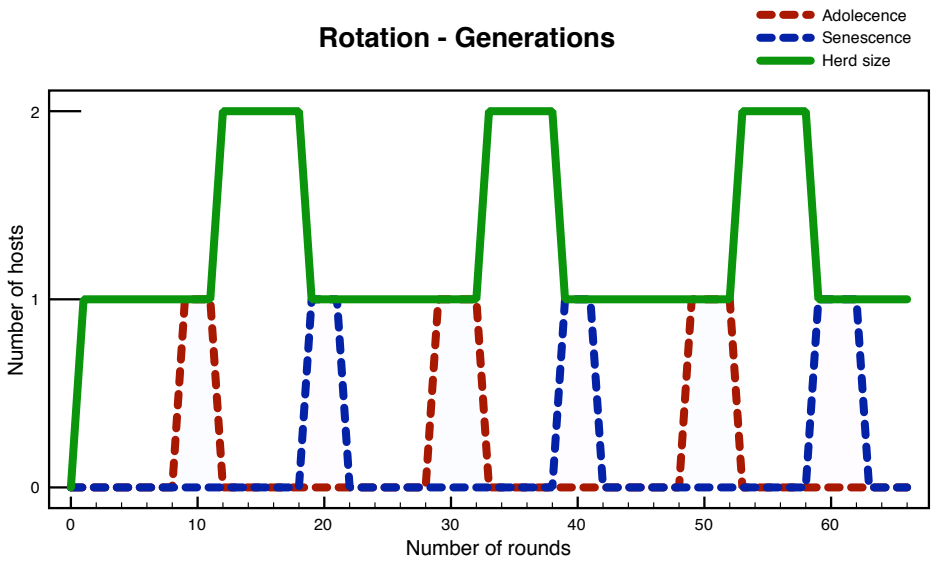


Figure 5.6: Demonstrating generations

Maintaining the senescence is important due to how the script chooses the next machine to boot. If a machine is powered off, but still not off, it may be started again if the time period is to large. Because of this the use of senescence is required.

Date	Host 1	Host 2	Host 3	Host 4	Host 5	Host 6	Host 7
1304820155 04:05:35	down	10	down	down	down	down	down
1304820140 04:05:20	down	10	down	down	down	down	down
1304820124 04:05:04	down	10	down	down	down	down	down
1304820108 04:05:48	down	10	down	down	down	down	down
1304820093 04:05:33	0	10	down	down	down	down	down
1304820077 04:05:17	0	10	down	down	down	down	down
1304820061 04:05:01	0	24	down	down	down	down	down
1304820046 04:05:46	0	N/A	down	down	down	down	down
1304820030 04:05:30	0	N/A	down	down	down	down	down
1304820014 04:05:14	0	N/A	down	down	down	down	down
1304819998 03:05:58	0	N/A	down	down	down	down	down
1304819983 03:05:43	0	down	down	down	down	down	down
1304819967 03:05:27	0	down	down	down	down	down	down
1304819951 03:05:11	0	down	down	down	down	down	down
1304819936 03:05:56	0	down	down	down	down	down	down
1304819920 03:05:40	0	down	down	down	down	down	down
1304819904 03:05:24	0	down	down	down	down	down	down
1304819889 03:05:09	0	down	down	down	down	down	down

Figure 5.7: Demonstrating generations in status webpage

5.7 Herd size equilibrium

Herd size equilibrium is a demonstration on how the scaling mechanism finds the ideal number of hosts or finding the herd size equilibrium. At this point the servers are operating at a load were it is not idling and not overworked.

At the start of the test the test environment consisted of a loadbalancer and one webserver running. By using httperf a fixed number of requests were sent to the server. The single webserver running started reporting high load, as seen in figure 5.8.

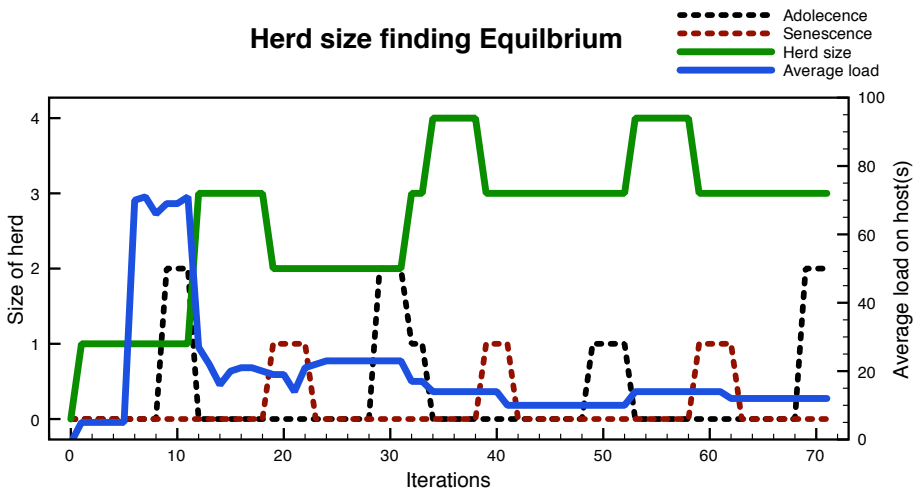


Figure 5.8: Graph demonstrating the scaling agent finding herd size equilibrium

As demonstrated in the table figure 5.4.2, the webserver scaled up to three machines. The average load on the system dropped, as the load was evenly distributed. At the next scale-down decision, the oldest webserver was powered off.

This again lead to an increase in load, and two more servers was started. This test was executed in the Amazon cloud, resulting on longer boot time than in a localized setup. The servers did not report online-status simultaneously, which explains the jagged rise in herd size.

After this, the load stabilizes and the oldest webserver is again powered off. However since the load is evenly balanced and not high enough to trigger the scaling mechanism, the servers have reached equilibrium. The change in herd size after this point is the rotation (aging) of the virtual machine.

5.8 Herd expulsion

Detecting the attack is a vital function of this scenario. Snort is an open-source intrusion detection solution and was chosen to act as the warning system. Snort analyzes network traffic and based on pre-set rules it can alert of attacks, scans or other anomalies. This alerts are then written to a log-file. The alert mainly exists of date, attack-information and priority. The higher the priority, larger the threat to the system.

There is a standard set of rules to use, and more are available through snort for an extra paid subscription. However in a testing environment using the rules that already exists may interfere with the test plan. Considering an nmap-scan is not uncommon and this could affect the results. Because of this custom rules were used to not only to set high priority, but also to reduce false positives (even though the may infact be true positives). The rule simply reacts to a certain command that is unlikely that others may use.

The first issue is creating a script that is able to read the alert-file, and parse the output to make a decision. The file must be read in a loop on an one second interval. Depending on the priority level of the attack one can choose to respond to the attack. The information gathered from the alert file is sent directly to a logging queue. After this message is sent the server sends out a message to a queue called warning. This queue is listened to by all servers and automatically blocks any IP-address that is sent through the use of IPtables.

All servers that block an IP-address sends a message to the syslogger-queue. This contains the hostname of the host doing the blocking, as well as which IP was blocked. This message is then received by the logging-agent, allowing for later inspection. A timestamp at the logger is used to show when the message was recieved, as well as a timestamp for when the block was issued on the host.

The Amazon SQS-messaging system uses an username/password to ensure that the message is not read/modified by other parties. However if an attacker has gained access to a system without it being detected by the IDS, the messaging system could be used to create a denial of service-attack by blocking other IPs. A prerequisite of this is that the attacker knows the addresses of the other webservers or other servers connected to the webservers.

A problem that appeared was that messages in the "warning"-queue stayed dormant. Usually SQS-messages are read once, then deleted. If the file is just read, the standard command is that the message should remain hidden for a certain timeperiod, then reappear. This would lead to a long period

for all machines to read the message. SQS offers the possibility to set the sleep time to 0, meaning it can be read at once. However the message was never deleted. This meant that the message was re-read and the drop-rules was continuously added.

This would not scale well so a script called rain was added. This is supposed to simulate the rain or air to remove the chemical signal of the communication. The script records a message at the same time as the other hosts, sleeps for 30 seconds, then deletes the message. A 30 second time-period should be sufficient for all hosts to read and block the attacked IP. Additionally all hosts store the variable and do not try to add a IP that is already added.

A problem that currently has not been solved is that the block is permanent. This means that if a webservers IP is blocked it is not removed before a manual action is performed. More information about this subject is in the discussion chapter.

5.8.1 Simulate attack

To test whether the solution was working a quick test was tested. By sending a secret signal ("installwindowsME") the attacked host detects the attack. By looking at the excerpt from the logfile beneath it is possible to see how the reaction progressed.

```
1  # Excerpt from recieve_log out-file
2  1303918009,ATTACK: webserver_1 (10.122.198.35) is attacked. \\
3  Someone attempted to install Windows ME. Priority: 11
4  1303918009,ATTACK: webserver_1 (10.122.198.35) is shutting down
5  1303918010,BLOCK:webserver_13,1303918043,block_ip:10.122.198.35
6  1303918010,BLOCK:webserver_4,1303918042,block_ip:10.122.198.35
7  1303918011,BLOCK:webserver_10,1303918044,block_ip:10.122.198.35
8  1303918011,BLOCK:webserver_2,1303918044,block_ip:10.122.198.35
9  1303918012,BLOCK:webserver_7,1303918045,block_ip:10.122.198.35
10 1303918012,BLOCK:webserver_12,1303918044,block_ip:10.122.198.35
11 1303918014,BLOCK:webserver_8,1303918046,block_ip:10.122.198.35
12 1303918015,BLOCK:webserver_9,1303918046,block_ip:10.122.198.35
13 1303918015,BLOCK:webserver_5,1303918049,block_ip:10.122.198.35
14 1303918016,BLOCK:webserver_15,1303918049,block_ip:10.122.198.35
15 1303918017,BLOCK:webserver_1,1303918051,block_ip:10.122.198.35
16 1303918018,BLOCK:webserver_14,1303918050,block_ip:10.122.198.35
17 1303918020,BLOCK:webserver_3,1303918054,block_ip:10.122.198.35
18 1303918025,BLOCK:webserver_6,1303918059,block_ip:10.122.198.35
19 1303918039,RAIN:rain has fallen, message, '10.122.198.35' is no more
```

The attack was registered at time 0. At the same time, the machine is powering off. The servers time are not completely in time-sync, so the timestamps

varies to some degree (+40 on the clients-side). However, the timestamps difference shows that all hosts blocked the IP within a 16 second time-period. After a grace-time of 30 seconds the "rain" has pored and the message is deleted.

The speed and scalability of the blocking-mechanism give a potential hacker the smallest time-window of creating havoc or gaining knowledge of the system. The risk of DOS-attacks is present with this soultion, but blocking the IP is just one type of response.

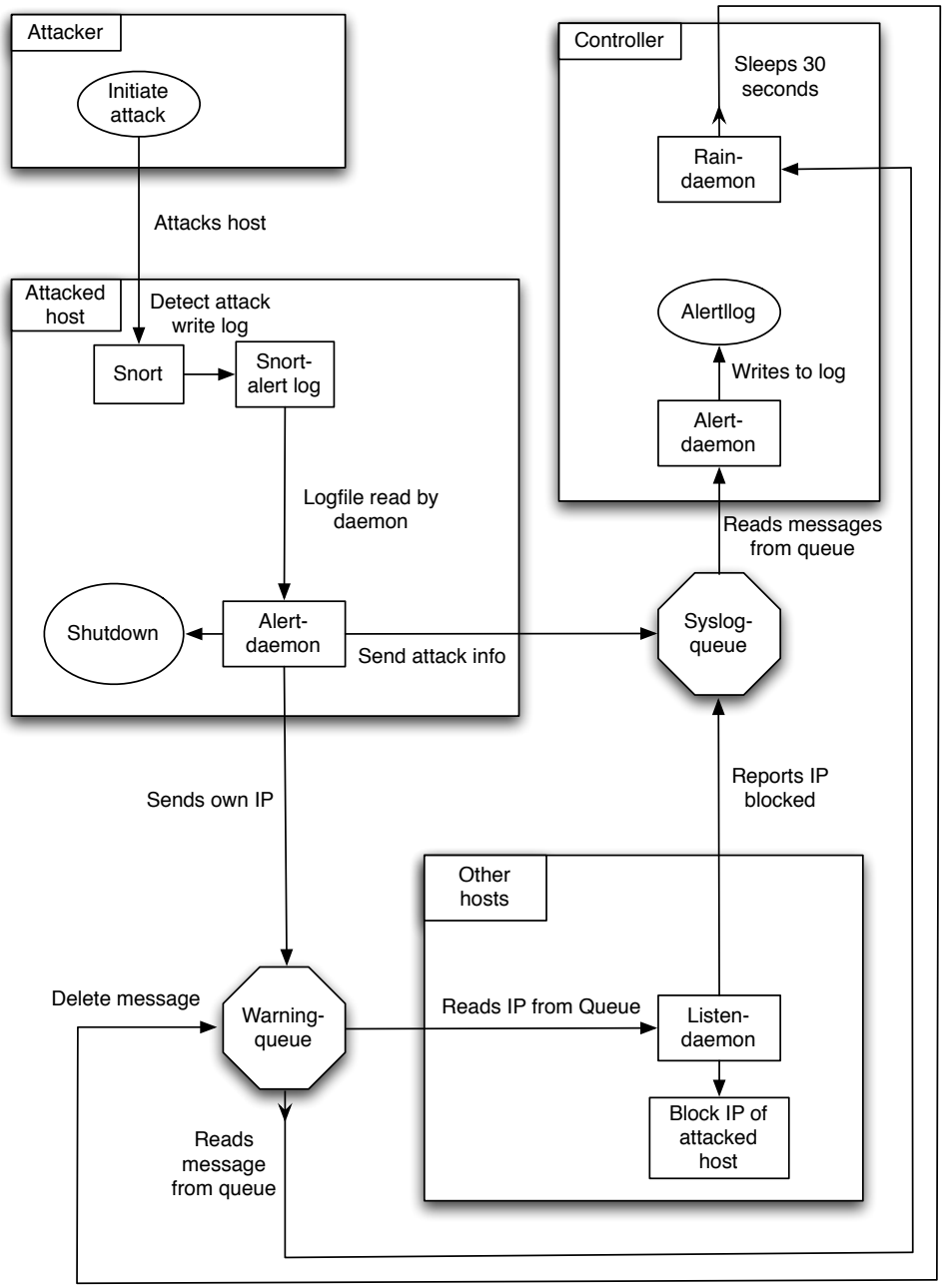


Figure 5.9: Attack warning implementation

Figure 5.9 demonstrates how the different scripts and sqs-queues are implemented.

5.9 Metabolism

Xen offers the possibility to change specifications of the virtual machine on the fly. This means that a server can get more CPUs or more memory to increase performance. It can also reduce the characteristics by lowering CPU or memory.

Amazon does not allow a change of hardware whilst running, but easily allows the change from one instance type to another. This means that if a server is having poor performance, the next server can get improved specifications. However with MLN this requires a rebuild of the project to allow a change in instance type.

Since the general build-time is around an hour, this is not a very rapid approach. Additionally the project has to be powered off while building, leading to down-time or a need for an alternative design. There is however a different approach to change instance type.

Building a MLN projects creates a set of bash-scripts that is used to start and stop the different instances. Inside these generated scripts there is a setting on what type of instance should be booted. By dynamically changing this type depending on load, the instance type can be changed without requiring a rebuild of the project.

This however leads to a challenge in platform. A change in instance type from small to large requires a 64-bit platform. The current platform has been 32-bit, and thus needs to be changed. More on switching between 32- vs 64-bit and the issues this caused is discussed later in the results chapter.

5.10 A complete test

5.10.1 Daemon issues

After starting a lengthy testing in the cloud a problem with status reliability became apparent. The problem consisted of a random crash of the status-reporting daemon. The crash caused a memory-fault and the software stopped reporting the load on the system. This was not a problem in the localized setup, but became apparent when using the Amazon cloud.

The crashes happened randomly, sometimes after 10 seconds, sometimes after 10 minutes. There seemed to be no pattern in how the crash happened and what caused it. The following steps were taken to attempt to solve the crashing issue:

- Stopping running processes (Apache, Snort, other daemons)
- Upgrading to a newer EC2-kernel module
- Moving from US-data center to EU-data center

After none of the following steps proved to have any effect on the problem, a closer look at the demonizing was necessary. It appeared that the daemon itself was causing the discrepancy. After switching from the perl-module Proc::Daemon to the linux program "daemon" the servers stopped crashing as much. Although the servers may at random times stop reporting their status, there was not enough time to debug this problem extensively.

This problem was not apparent before a large number of hosts were used. Two main points can be learned from this. Firstly it proves that the use of rotation has in fact a purpose as software has bugs and tend to crash. Although in this scenario it almost ruined the test, it shows an interesting prospect. Secondly it proves that the current script has too poor control on the data received, and a later version must improve this further.

5.10.2 Thresholds

The threshold values used to scale needs to be set accordingly to the scenarios. By installing and running the Snort IDS the load on the system will increase slightly. This will also be the case when using a server with different specifications. A small server will have generally higher load when running the same services than a larger server.

Because of this a test of the "average" load on the system without any external load was tested. To simplify the decision-making process only CPU is used in this scenario. The average total load on the system was 5 %. This means that if the server is running under 5 percent, it is idling.

The following metrics where obtained from httperf. At 15 requests per second the webserver is at it's threshold. Any more requests per second will result in timeouts. The CPU-load at this many requests was 60 percent. This gives the following thresholds boundaries listed in table 5.2.

Increase cycle	
Load	**Percentage change**
40-100 %	50 increase %
25-40 %	25 increase %
25-0 %	10 increase %
Decrease cycle	
25-100 %	10 decrease %
0-25 %	25 decrease %

Table 5.2: Herd size scaling thresholds

5.10.3 Offspring mortality

In certain scenarios the server does not behave as aspected at boot. The server may not complete it's boot-circle, it could crash or other factors that inflict the server. The server is suppose to send a message after all the runlevel-commands have been issued. If the server hangs on a run-level step, the machine will never send its IP to the scaling script. This will again lead to that the machine will never exit the "adolecence"-pool.

Due to time limitations it was not possible to implement a solution to solve this problem in real time. Because the machine is in the pool, it will not affect the results in terms of load. Additionally the machine will never be stopped or started again. A simple timeout in the pregnancy pool where a machine is killed if it is not up after a predefined time period.

This is also reflected in nature. Unfortunately not all offspring survive their adolescence, and are succumbed to sickness, predators or other factors. For instance, an american female robin may produce a dozen offspring in two or three clutches during the nesting season, and it is common for only five or six to survive. It is nature's way of maintaining populations that the environment can support.

5.10.4 Creating dynamic load

Originally the graph from finn labs was suppose to include data values which could be used to replicate the real life scenario. Unfortunately, the data was not available at the time of the experiment so a recreation of the data based on the image supplied was done. It is not an exact replica, but it gives the possibility to show how this script would perform in a real-life scenario as close as possible.

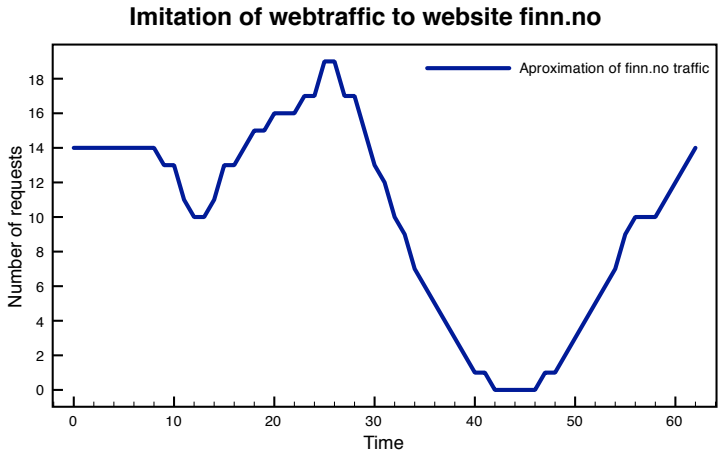


Figure 5.10: Recreation - number of requests to finn.no

5.11 Cloud to localized virtual setup

Due to the frequent crashes and general instability in the Amazon cloud a shift to a localized setup was necessary. However, one of the key factors deciding towards a cloud based was cost. Getting hardware with enough power to be able to recreate a large scenario would be to expensive.

A server with a quad-core processor was obtained to try to recreate a large installation scenario as closely as possible. By doing this, a four virtual machine setup would mean that each server has one virtual CPU each. However, a scaling beyond four servers would not drastically improve overall performance. This server was installed and configured with Xen and MLN.

Upgrading Xen

During the implementation phase, an update to the Xen hypervisor was made available. This was installed using apt-get. However the installation process failed and it was impossible to boot onto the Xen-kernel. The files was retrieved using the standard Debian-kernel and the server was reinstalled.

5.11.1 Simulating an attack

The localized setup required a change in network topology. The cause of this was mainly due to lack of available IP-adresses on the network, but the loadbalancer also needs to function as a gateway. This is easily configured with Xen, but the throughput on the gateway will affect the performance of the webservers since they are all running on the same physical components.

Another issue that appeared in a localized setup was Snort seemed unable to read network traffic reliably. This is most likely due to the NATing used by Xen. When testing without NAT and in the Amazon cloud, Snort was able to listen to network traffic and detect the attack without problems. Since Snort was unable to detect our specially constructed attack, another approach was needed.

By collecting an alert-log sample from earlier it was possible to simply copy this into a running VM. The alert-daemon would then read the file, and initiate the attack warning cycle as under normal circumstances. Although this breaks with the goal of simulating real life scenarios, the goal of this scenario is to demonstrate the principle, not demonstrate the effectiveness of Snort in a NAT network.

```
# Custom snort attack
[**] [1:1337133712:1] Trying to install_windows PANIC [**]
[Classification: Someone attempted to install Windows ME] [Priority: 11]
05/06-15:25:41.506575 128.39.28.178:49963 -> 10.241.99.48:80
TCP TTL:50 TOS:0x0 ID:29378 IpLen:20 DgmLen:547 DF
***AP*** Seq: 0xB8128E5A Ack: 0x726EEDBA Win: 0x8218 TcpLen: 32
TCP Options (3) => NOP NOP TS: 401059347 126539
```

By watching the dips in number of hosts at "" and "" it demonstrates the principle of attack warning with herd expulsion. Host_1 is under attack, snort detects the attack and the alert-daemon sends out a warning message. After this message is sent, the machine powers off. The load on the other machines increases, recognized by a spike in load.

The increased load leads to a more aggressive scaling decision, starting more machines to handle the increase in traffic. The load generated reached almost 60 percent, which is close to the performance limit of the webserver. However the amount of webservers increase and the load is evenly distributed.

By collecting the number of requests at the webserver it is possible to see how many requests are registered at the balancer, see figure 5.11. Although this information can be regarded as redundant since it already exists from our test-script. However it works as a control mechanism, ensuring that the number of requests planned and executed are similar. Additionally it works as a method to ensure that no other requests were sent to the server and thus having skewed the results.

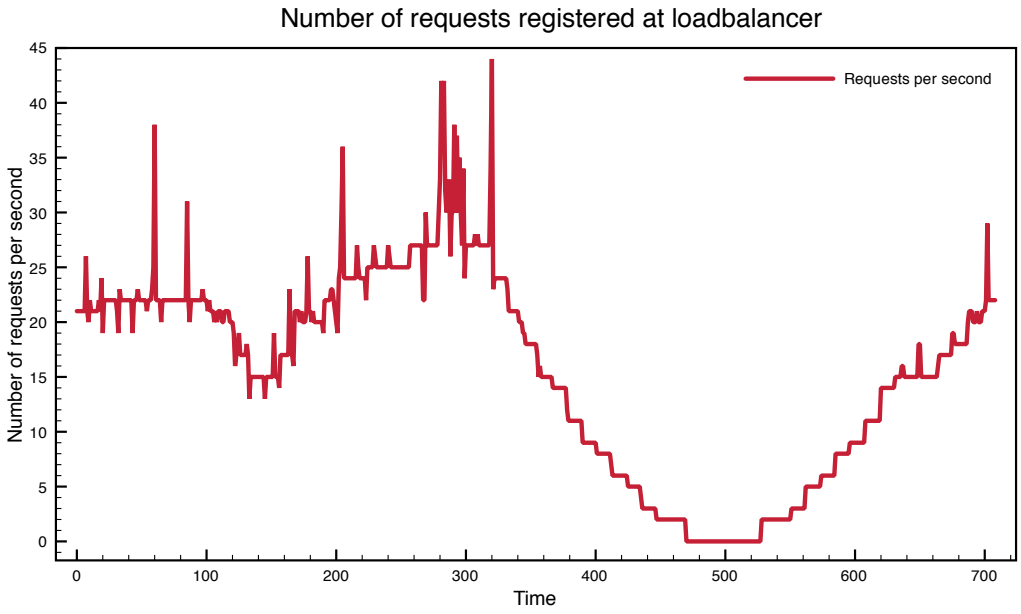


Figure 5.11: Incoming requests registered at loadbalancer

The test started with only one server running, and the scaling script should scale to find the equilibrium between servers and load. The general high load at the beginning is caused by all the load being sent to one server. As more servers are started and added to the pool, the load stabilizes around 25-30 percent.

During the simulated night-time the number of requests and load goes down. The CPU-load does not however reach zero due to the standard load on the system (for example snort). The increase in traffic in the morning shows that the scaling mechanism has faults. There is a large spike in load, close to the limit of the webserver.

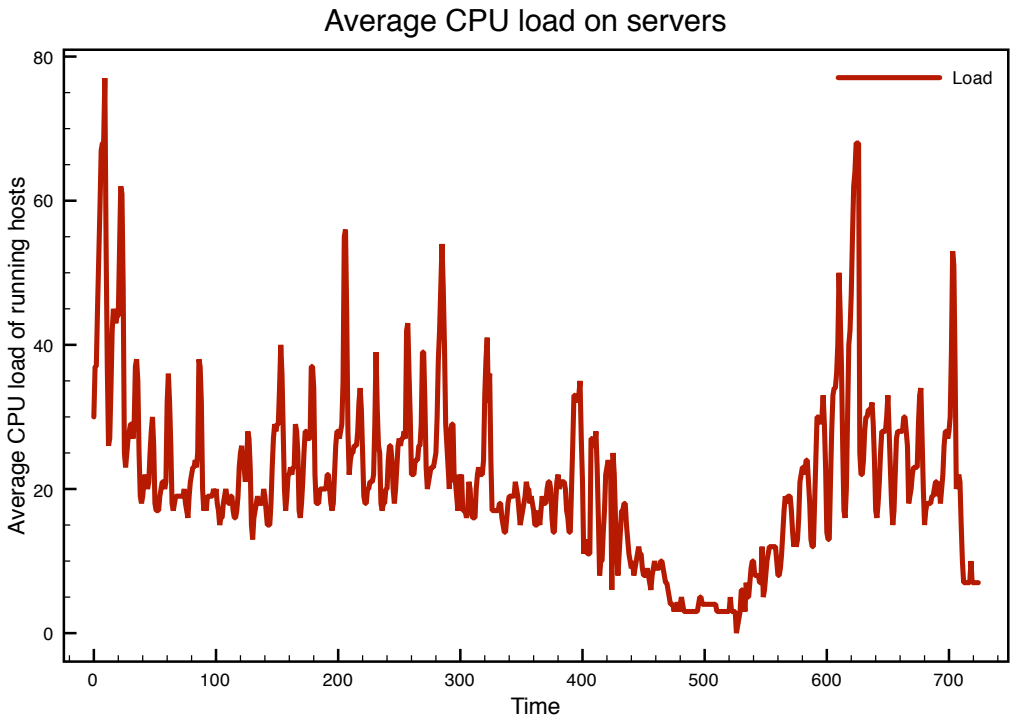


Figure 5.12: Percentage load on running hosts

Watching the number of servers in figure 5.12, it tells different stories at different times. Firstly it shows how the scaling script works to scale the number of hosts. It also shows the generations, with old machines being powered off. This represents as a potential problem in our scenario, scaling down due to age regardless of the load on the system.

Two times during the test a machine was attacked, as seen marked in red circles in figure 5.13. This demonstrates that the server-count drops, since the machine powers off as a part of the defense mechanism. The load on the other machines increases, thus resulting in more machines being started.

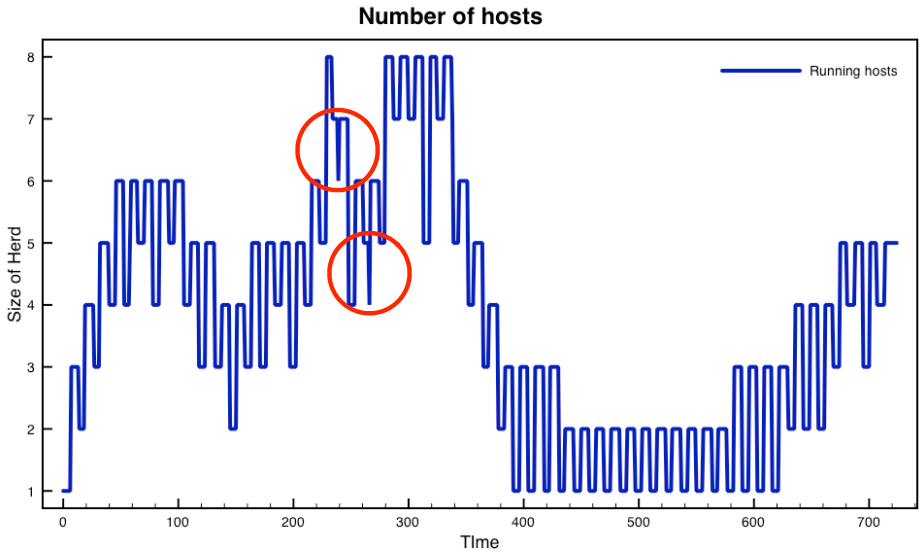


Figure 5.13: Number of hosts running

In the time of the day when the load was at its highest, the maximum of hosts were up. The scaling script has control over the number of hosts running, and does not try to start more machines. The aging-process is still present, and the oldest machines are replaced by new ones.

As the traffic drops the number of servers drops accordingly. It scales directly from eight machines down to five. This demonstrates the principle of reducing the resource consumption in virtual machines. Since the load went down, the number of servers followed. As the load is lowered down to zero requests per second, the number of webservers go down to the minimum level.

The generations are more visible in a low amount of servers.

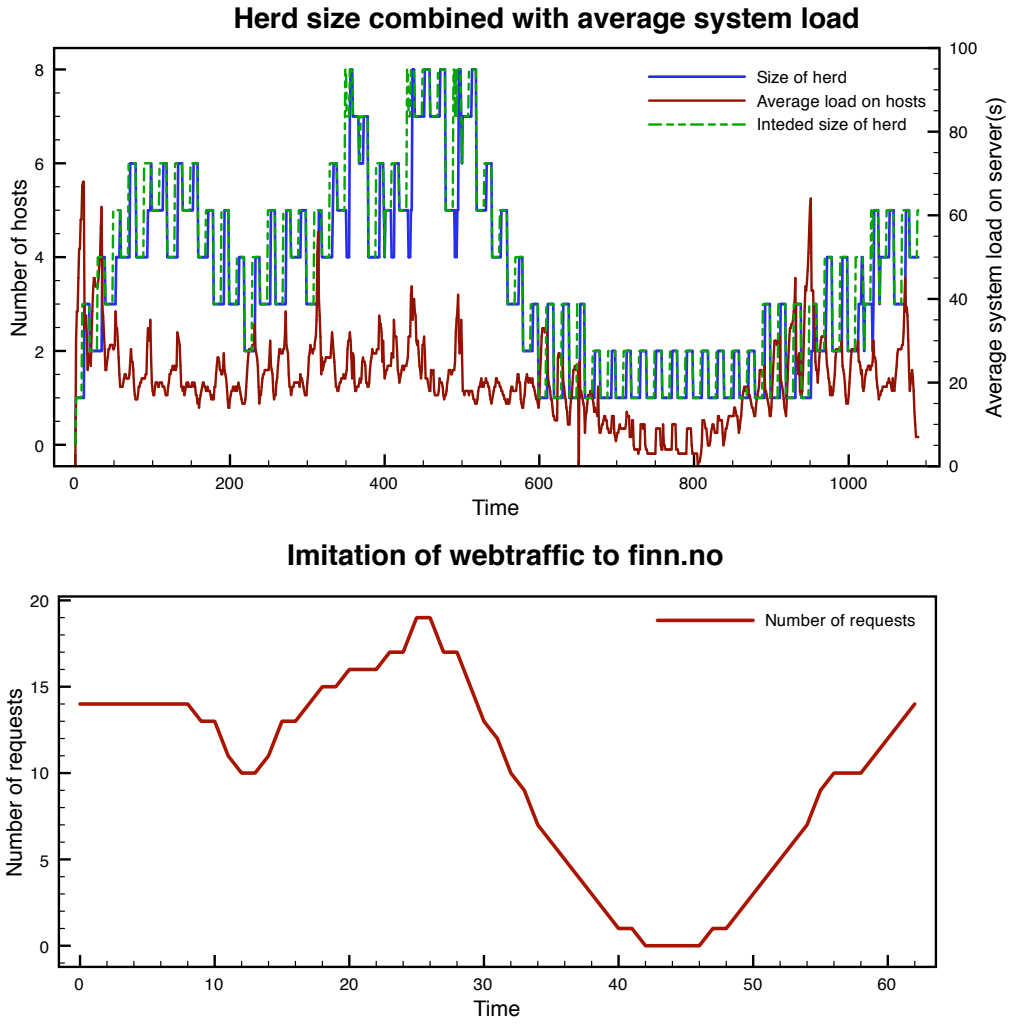


Figure 5.14: Number of hosts running

The intended number of servers is when the scaling decision has been made, but the host has not yet been added to the pool. This has earlier been called adolescence or pregnancy period. The reason why the senescence is not included in this graph is that the machine is automatically removed from the pool when the scaling decision is made.

Since this was a localized setup with a minimal installation, booting up a VM is reasonably quick. By comparison booting up a machine in amazon can take up to a minute to complete the boot-cycle.

Chapter 6

Discussion

This chapter will discuss the different aspects of this project, in terms of results, approach, design and process. Finally a suggested answer to the initial problem statements will be proposed.

6.1 Biology as a design process

This section will cover how using biology as a design process or methodology. It will present different strengths and weakness of this approach and the overall goal of using biology.

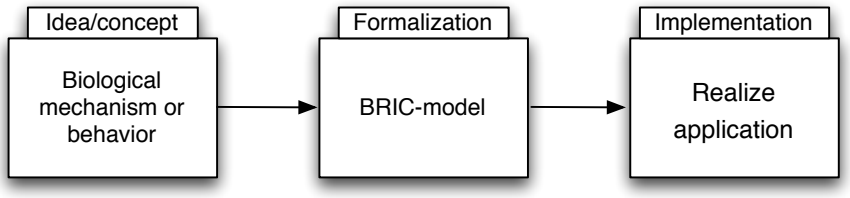


Figure 6.1: Using biology as a process

By watching figure 6.1 it demonstrates the design process for Q1 and Q2. It is worth noting that this method includes the implementation, but not all models were in fact implemented.

6.1.1 Biology strengths

Using biology as a design process has several positive traits. Approximately the number of species on the earth is between 5-30 million[58]. Each animal has different characteristics and has adapted to their different environments. Animals that do not adapt will become extinct.

The continuos adaption towards the changes in environment is one of the single greatest traits in biology. Adapting herd and physical size, traits or defense mechanisms are just some examples. The large amount of mechanisms, behaviors and different animals represent an untapped resource.

Strengths in simplicity

The use of analogies in teaching is a well-known method to teach complex scientific subjects[34]. Using the familiar, something everyone knows, it is much easier to relate. By relating to the information, the listener becomes a larger part of the learning process.

Biology is something that is thought from very low age in school. It's showed in the news and on the television. By comparison, computer science is much more limited in its reach to the general public. Everyone has a computer, but not many understands the principle of a datacenter, virtualization or different security mechanisms. By the use biology, this terms can now be explained in a way that a non-computer educated person can understand.

6.1.2 Biology weakness

To be able to use biology to explain complex data systems or mechanisms requires knowledge over two very different and very complex fields of study (Biology and System Administration). The two fields tend not to cross, and the amount of people with knowledge of both system administration and biology is very limited.

Learning the terms, expressions and methods used in biology is a time consuming process. It requires large amounts of reading and investigating to find the correct meaning. It cannot be compared to learning a new programming language or mastering a new type of technology. The difference in fields is perhaps to great.

As an example of this challenge, the expression "a lizard loses tail". This statement is true for some types of snakes and lizards. It is a secondary defense mechanism to avoid consumption by a predator. However finding the different elements is a difficult process. Learning about the different animals requires time, however the mechanism itself is not the most comprehensive.

Learning about different types of animal defense mechanisms was an interesting process. By simple reading about the subject, ideas came on how the different methods could be used in System Administration. However the language used is complex and requires a degree of biological understanding.

Limitations

Although biology has proved to be an interesting approach to manage virtual machines, it has some limitations. Biology-, by concept has faults. Animals that are not able to adapt become extinct. This can be natural or caused by human factors.

Considering a service that uses biological methods to manage it's virtual machines. If there was no load over the weekend, by default all servers would die over the weekend. This would be impossible in a production environment.

Examples of limitations

In cases of high load on a system-, it is a disadvantage to have generations. A low amount of servers and high load means that if one server is under load, two will be started. Then at the next generation-, one will be shut down regardless of the load. In biology this makes sense, more food does not increase your ability to live longer. However the limitations in biology can be avoided in system administration.

By simply setting a rule in the scaling script of not allowing a machine to be powered off if the load is greater than a pre-set value, it is possible to improve scalability drastically. Not only will the number of servers grow faster, since no machines will be powered off, powering off a server will also increase the load on the other servers resulting in potential errors on the client side.

6.1.3 The goal of using biology as an approach

Defining a clear goal of using biology as an approach or a process is difficult. Is it a method to develop new types of behavior or mechanisms in system administration. Is it a method to explain an existing technology or platform. Or is it a method to explain and design a a perfect system (animal).

Defining the perfect animal/server

Defining a perfect animal or server is an herculean task, and in practice an impossible task. The larger number of different species are a result of evolution and adaption to the environment. An elephant has adapted to have the best protection and method to obtain food and survive. However it would not survive in a desert. The perfect animal is a product of it's environment.

The same is true with servers. One server does not fit everyones budget or requirements. Not all functionality is needed and not all servers are secured

in the same way. Defining a perfect animal or a perfect server is not possible. Different needs and requirements makes such a definition unrealistic.

6.2 Translating biology to automated management processes

By using the models created in Q1-, a set of implementations was achieved in Q2. The models were first tested separately-, where the individual functionality for each different model was tested. Then a larger test covering all the implemented models were tested.

A larger test which combined different models was also executed. In this test the number of servers would dynamically scale depending on the system load. If a server was attacked, the machine would send a warning to all other hosts, causing the attacked host to be blocked. The attacked server would power off-, the increased load on the other servers would make the scaling script replace the attacked webserver. The servers would also be continuously rotated, meaning the oldest machine would get replaced at a fixed interval.

Although simplistic-, this demonstrates how server behavior can be automated with just a set of scripts. Dynamic scaling of servers, fault-correction and security are handled automatically without manual interference from a system administrator. The system administrators role is defining a set of rules for the scripts. This is for example maximum herd size, ferocity of scaling, frequency of scaling and etc.

This is means that the system administrator can focus on how to provide better service, instead of performing mundane manual tasks.

If a larger localized data center was available, a more comprehensive approach would be possible. As seen in figure 5.2, being able to effectively power on/off physical machines would further demonstrate the potential savings of more effective data center usage.

6.2.1 Supplemental results

Supplemental results covers results in the form of data or different metrics that could augment the results. In the process of planning, testing and analysis certain gaps in the experiment appear. Whether this is different types of data/metrics, supplemental data or change in method. Problems with hardware or software may lead to a change in the approach, and by extension making the initial data collecting useless.

An example of this type of issue appeared at a late stage in testing. Due to the issues in the Amazon cloud, a switch to a localized setup was required. However the number running hosts had to be considerably lower in the localized setup than in the cloud. This was due to the hardware limitations of the available servers. There were only four cores in the server, and scaling beyond many more than four servers meant they had to start competing for resources. This meant that after four servers were running, adding more would not improve performance.

Due to this, there was little to no point in addressing timeout and error from httperf. If the servers would scale, it would be more important to look at the performance figures from httperf to ensure that the number of errors was low. Without this, we have no statistical proof that the scaling method ensures quality control, but the general load on the system is recorded serving only as an estimate.

A larger focus on resource consumption would also have supplemented the results greatly. With the current version of the scaling script, machines are turned on/off regardless of how old they are. In Amazon, one pays for the number of hours used, so even if a machine powers off after five minutes, you still pay for the complete hour. Introducing a mechanism that manages this would greatly increase the financial strength of the scaling script.

This type of change would not require an abandonment from biological thinking. This could be introduced as fat-reservers, allowing the body to continue living for longer periods without eating. This is just another example on how biological thinking can represent and potentially solve problems or issues that appear.

6.2.2 Reproducibility

The results to Q1 can be considered to be quite reproducible-, in terms of available information and available tools. The results from Q2 however are more complex, and represents a larger variety in terms of tools and platform.

Using the cloud allows to recreate earlier scenarios with almost identical performance. If using a localized hardware based setup-, the performance and reproducibility would rely on the hardware at hand. Certain types of hardware are not available everywhere in the world-, and thus a scenario requiring the exact same hardware would be difficult.

All software used, with the exception of Amazon SQS is free and open-source. The source code of the different scripts produced in this thesis will

also be made available free and open-source. By using open-source software there will be a lower burden on future research.

The projects used in this experiment will be deleted at the end of this thesis due to the use of borrowed hardware. However the mln-project files and main image will be stored to allow to continue the research.

The graph obtained from finn.no was given under the condition it would not be used to anything else than intended. Using this graph is only allowed under the direct approval from finn.no.

This thesis has been heavily dependent on Amazon and the services they provide. Although the current financial and vision of Amazon is solid, the services may be discontinued after a longer time period. The services may be replaced, and the libraries used in this thesis may be obsolete. This is one of the greater challenges by depending to heavily on a single provider.

6.2.3 Machine uniqueness

The virtual machines names and characteristics are predefined. Only the number of hosts can be changed in the scenarios in this thesis. By doing this it is impossible to know how the herd has grown and changed over a period of time. The servers only grow by numbers, never by size.

The lack of lineage and not having an evolutionary aspect limits the effectiveness of the approach. If the herd size reaches it's maximal size, there is no more room for the herd to grow. The other way also limits the possibilities, a herd can decrease to the population of one; but never go extinct. Modifying the characteristics of the servers can change this.

6.2.4 Attack warning - Temporal boundaries of behavior

In the attack warning scenario it is demonstrated how a machine could be blocked from the rest of the herd if it got an illness. As an idea it worked-, but it fails to address some issues. If a machine is blocked-, there is no mechanism to unblock it. This would perhaps not be a problem when using Amazon because of the large amounts of IP-addresses. However in the case of a localized setup, a new machine will be likely to obtain the IP of an old machine that is blocked.

In biology different methods are used to "end" communication. With sound

it is only recognizable when it is made. Chemical responses are diluted over time. The messages are subjected to the change of air, wind and rain. The messages end-, but the animals response persevere. Through the use of hormones animals can adjust to the different types of situation that can occur.

Hormones does not however stay forever. The time a substance is active is called "The biological half-life". In case of adrenalin this is only two minutes[24]. By looking into adrenaline in more detail we can see that it causes physical changes. It increases the heart rate, make blood vessels in the legs wider and can lower the immune system. This can be compared to using lots of resources to combat an attack at the cost of idle cpu, thus potentially limiting other important functions (services).

However when the adrenaline is gone from the animals system everything is back to normal. The animal then senses nothing different from the earlier state it was in before the attack.

Working further with hormones as a base is an interesting prospect, but due to time limitations it is not possible in this time frame. However looking at the resources used by for example IDS or IPS it is quite similar to that of hormones. Looking at the connection between communication channel, limiters in terms of air, wind or rain, and finally how the server should respond has a lot of potential.

6.2.5 Using Amazon SQS as a messaging tool

Amazon SQS has been an interesting tool to work with. It has lots of possibilities in terms of flexibility and scalability. However as a messaging platform it has proved hard to work with in this type of environment. Using the tool differently from what was intended required a lot of time to script around issues. The amount of work that was put into using SQS as a messaging system was not proportional to the results.

There are still unresolved issues with servers that suddenly stops reporting their status. This leads to inaccurate scaling decisions .There are no mechanisms for sending widespread messages, so it had to be tuned to do this task.

Finding an alternative messaging platform with the same capabilities as SQS, but none of its drawbacks is something that should be a future priority. It should be noted that the instability issues regarding SQS has not been fully investigated. It is difficult to know whether it is the perl-module or the SQS design that cause the problems.

6.2.6 Using a localized versus a cloud based platform

Choosing the right platform for this thesis was difficult. To achieve the level of scaling desired required either powerful hardware, or renting the hardware through the Amazon cloud. Easily available and affordable the Amazon cloud appeared to be the most prominent solution. MLN offered the possibility to test locally, and by simply applying the ec2-plugin[11] to upload the project to the Amazon cloud.

However several technical challenges appeared, all of which are thoroughly explained in the results chapter. Examples are trouble with iptables and general instability of the servers. The support for Amazon was first offered in 2009[11]. The modules used by MLN was last changed in 2006, and has been suggested by Amazon to be considered as to old and flawed.

In this thesis a Debian 5 .0 lenny-image was used as a starting-point for all servers. All configurations, installations and tweaks was done on these images. Using Squeeze was not possible with MLN and Amazon at the the time of testing, so it was not possible to switch servers.

If a similar testing was to be attempted, a couple of guidelines in the chose of platform should be set. Firstly, to the furthest extend try to get a localized setup. Testing and configuring is less time consuming locally than in the cloud. A typical Amazon image takes 40 minutes to upload. If the image was misconfigured, it would need to be uploaded again. This is a typical painstaking task, and steals too much testing time.

By comparison, a localized setup build is dependent much more on disk performance, number of machines and size of image. In a typical scenario with five hosts, a build seldom took more than five minutes from build to startup. This meant that testing was quick and agile, allowing easily for experimentation.

Another factor to consider-, is the flexibility of MLN in a local setup. MLN has recently started to offer support to VMWare ESX. Previously + it supports Xen and User-mode linux. In case of a platform instability or incompatibility, it is possible to change the underlying platform without a large change in the project file or images. This would typically not be possible in the Amazon cloud.

6.3 Q1 - alternative approaches

The current approach is an two part process. The first part, consisting of Q1 is mainly an theoretical and results in a abstract model. However the goal of the problem statement was to simplify the conceptual complexity. This was achieved through the process of creating models, but is currently not verified by any external controller.

Using a survey, either in the form of a questionnaire or a interview, make it possible possible to further demonstrate if the desired goal was achieved. By explaining complex systems using biology versus using normal technical skills it would be possible to test if the explanation was successful. Alternatively a set of tests on explaining different types of pre-set systems could be used.

6.4 Q2 - alternative approaches

Compare scaling script to Amazon

As a part of the Amazon cloud package, amazon supports autoscaling based on the system load. Comparing the scaling tool created in this thesis with that provided by Amazon could be an interesting task. How well would it scale, would it scale more or less aggressively?

6.4.1 Introduce evolution

The theory evolution is defined as the change over time in one or more inherited traits found in populations of organisms[36]. Nothing makes sense in biology without an evolutionary view[23].

So far computers have been managed through the thought of intelligent design. Humans, i.e.. god, have set the rules, behavior and characteristics. If servers behaved as computers and were able to adapt to a change in food or environment, servers would be autonomic.

6.4.2 Towards autonomic computing and self-management

The goal to reduce management is also a step towards autonomic computing. Autonomic refers to the self-managing characteristics of distributed computing resources, adapting to unpredictable changes while hiding intrinsic complexity to operators and users[43]. A key part of autonomic computing is self management. Especially the following:

- Self-Configuration
- Self-Healing

- Self-Optimization
- Self-Protection

By comparing the implementations toward that of the minimum required characteristics of autonomic computing, it is possible to suggest that it is a step towards self management. The minimum required properties are automatic, adaptive and aware[32]. Automatic means it should be an automatic function, without the need for manual interference. Adaptive means it must be able to change its operation. Aware means that the system must be able to monitor its operational context as well as it's internal state.

	Self -			
Concept	**Configuration**	**Healing**	**Optimization**	**Protection**
Attack warning	no	yes	no	yes
Herd size	yes	yes	yes	yes
Loose tail	yes	no	no	yes
Aging	yes	yes	yes	no

Table 6.1: Table of comparison

Without the exception of "herd-size"-, no concept fully qualifies all requirements for autonomic computing. However by all the tools working together, all requirements are fulfilled and can be considered autonomic[47]. Different components working towards a common goal.

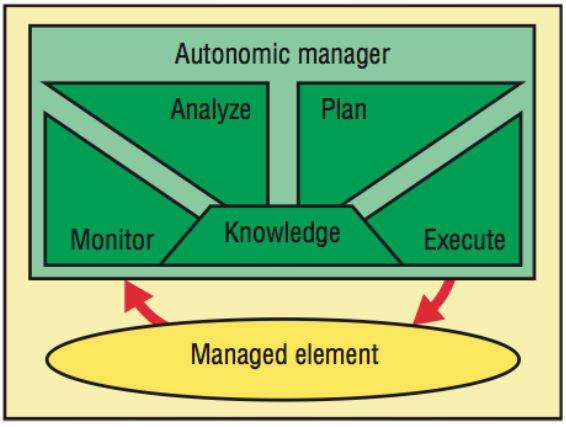


Figure 6.2: The structure of an autonomic element[43]

Figure 6.2 demonstrates the four different steps in an autonomic cycle. The autonomic manager monitors the observed element, and the analyzes the

collected data. The next step is planning possible solutions and actions based on its knowledge base. Finally it executes its plan to change the behavior of the managed element.

6.5 Technical complexity

The goal of creating a solution that utilized exiting tools was done as far as possible. However not everything was possible by using the standard tools, so some tasks required scripting to achieve the desired outcome. The herd-size implementation represented large amounts of work in terms of planning and execution, but required less than a thousand lines of coding to achieve successful scaling.

This clearly demonstrates the strengths of utilizing exiting tools as the foundation. Using simple scripting it is possible to orchestrate the different tools to work together towards a common goal if it is clearly defined by its designer. The different solutions such as starting and stopping a virtual machine already exists in MLN, so designing a scaling process requires small amounts of code.

With the exception of MLN and SQS, all tools are used by system administrators every day all over the world. Amazon SQS and MLN is more of a niche solution, which is not as widely used as Xen or for example using iptables as a firewall.

The libraries used in Perl are pretty standard, with the exception of sqs-simple, sys-statistics and the demonizing tool. The tools are still being supported and updated by its authors. All versions of the software has been updated within the last three months.

6.5.1 The cost of birth and death

By looking at figure 5.6 one can see the time a machine uses to boot or shutdown. This time period is quite large and uses a lot of resources. This shows how costly the process of rotation is and the cost of booting and stopping. The balance between aggressive scaling and the age of machine must be compared to that of the actual cost of booting. If a machine uses half of its life booting, it is an unfortunate use of resources.

6.6 Cloud-based challenges

In this thesis Amazon EC2 instances have been used to provide a more scalable setup at a reasonable price. The simple pricing model and large amounts

of available resources was a promising prospect. However the cloud-model as it is today is not without fault. The large Amazon downtime of 21st of April 2011 where the datacenter in North Virginia made several large sites like reddit, expono and many others unavailable.[22]

This does not mean that the cloud or Amazon is at fault as a technology or as a provider. It does however mean that the setup needs to be designed with the possibility of a fault in a single datacenter. Amazon provides data centers in Asia, Europe and in the US. The possibility of moving the your server park from one data center to another must be a possibility. Another possible solution is to use the cloud as a redundancy mechanisms to allow for easy upscaling of resources and to work as a failover mechanism.

6.6.1 Cloud and Politics

The whistleblower website Wikileaks became famous for publishing large amount of diplomatic correspondence obtained illegally from the US government[56]. Wikileaks had their servers hosted at Amazon at the time of the publishing. After a series of attacks on the website and protests from US officials Amazon stopped hosting Wikileaks. Amazon later released a statement[4] where the reasoning for the drop was presented. It was not due to pressure from US officials, but rather that Wikileaks did not have ownership of the files in question. This was a breach of the terms of service and thus the contract was terminated.

Regardless of the reasoning on why Amazon stopped hosting Wikileaks it shows how much of a change in the paradigm of web hosting. Instead of buying a piece of hardware and being responsible for the content itself, the responsibility is now that of the cloud provider. If the business or organization does not comply with the terms of service the provider can stop hosting your website. Considering the continuos change in the worlds diplomatic climate, the political influence on providers may increase over time.

The use of politics to influence the powers that be in the world of computing is not uncommon. The use of political muscle used in the case of The Pirate Bay[57][53] is one example of a countries desire to protect it's national income. The multi-billion industry that is piracy inflicts large amount of loss on the movie industry, thus inadvertently damage the United States economy through loss of income[42].

Another possibility is that a server is used by an political fraction or organization that is deemed undesirable for the hosting country. Other cases may be newspapers that obtain information from classified sources. The balance between freedom of speech and the Amazon terms of service may

result in an undesirable environment for potential costumers.

In this cases the Amazon cloud is to be compared much more to that of an ISP under control by a potential foreign country. The content on the webservers will be under the cloud providers jurisdiction. By looking at this aspects, the use of a single provider of cloud computing is to risky for long term use and the clients need to plan and adapt accordingly.

6.7 The learning experience

When starting on this project, I initially started with reading about different types of biological mechanisms and animal behavior. Through this process, I started thinking; "How can this be used in System Administration". Of twenty ideas, maybe only five could be turned into models. Of those five models, only one or two could successfully be implemented. Not every model is suited for use in real life.

This process was time consuming and required a creative way of thinking. This is perhaps the most beneficial result, making me always think how something can be used or further improved.

6.8 Affected fields

This section will address who and which fields this thesis results will affect.

Improved utilizing of resources

Using less computing resources is a vital part of green-IT[50]. Maximizing the use of running hardware so no unnecessary other physical servers are running or acquired. Actively using scaling to match the traffic allows for a more efficient use of computer resources and can therefore potentially lower the environmental footprint.

Virtualization has also been introduced as an important contributor to green computing. Virtualization also enables data centers to consolidate their physical server infrastructure by hosting multiple virtual servers on a smaller number of more powerful servers, resulting in less used electricity and a simplified data center. Besides getting better hardware usage, virtualization reduces data center floor space, makes better use of computing power, and reduces the data centers energy demands[50].

6.8.1 Teaching and training

Using biology as a metaphor, complex systems and programs can be more easily explained. This is useful in typical training or teaching scenarios. Having an alternative

Non-technical personnel

The use of a well known and easy to explain concept (biology) it is also possible to explain complex computer systems to non-technical personnel. This is probably on of the greatest feats of this methodology. This can be used by system administrators or teachers to demonstrate why a feature is important.

As an example, considering herd size and scaling. When implementing a system, there will be decisions of a technical character that is of vital importance. Different types of solutions are discussed in terms of features and the cost of buying, implementing and managing. Typically the customer wants a solution as advanced and secure as possible, at the lowest price. Often this is a contradiction, since one possible excludes the other.

When discussing why a scaling mechanism is of great importance, the system administrator can refer to how the size of a herd. Then one could explain how the herd grows and shrinks depending on how much food is available. If the size of the herd could not grow, food would not get eaten and this would be a loss to the company. If the size was to large, the cost of having a large herd with no food is also not be beneficiary.

6.8.2 Reduce mundane management

Reducing overall complexity and automating management processes reduces the overall workload on the system administrator. This does however not meant that the goal of this thesis is to remove the role of the system administrator. The goal is to get the system administrator to focus on the main tasks, not on mundane and repetitive tasks as suggested by Mark Burgess[13].

Just like comparing an wood burning stove to a modern stove, the modern stove requires less manual interference to maintain temperature. This does not remove the role of the cook, but it allows the cook to focus on more important tasks. The goal of manually configuring and managing each individual machines behavior will soon be regarded as old fashion as cooking on a wood burning stove. It has its charming features, but as an everyday task, one would prefer the modern stove.

6.9 Future work

Creating and implementing a larger set of behaviors and mechanisms is the most predominant task. Due to the short time span in this thesis, only a select few models could be implemented. The general goal was to demonstrate how the different models could cover all focus-areas.

To be able to successfully introduce the tools created into a production environment, demonstrating their effectiveness is also highly desirable. In this scenario a more general approach would be possible. A larger focus on potential savings in terms of higher utilization of server and testing security mechanisms against real threats. It would also be possible to see if by introducing aging-, is reliability improved?

Another possibility is further developing the method to use biology to explain complex management processes. This can be useful for system administrators, teachers and managers that need to understand or explain complex management processes.

6.10 List of scripts

This is a table presenting the different scripts created in this thesis along with a description of its features.

Scripts		
Scenario	**Name**	**Description**
Attack warning	alert_daemon.pl	Daemon to read, analyze and send warning of attack. Initiates shutdown procedure of attacked virtual machine
	listen_daemon.pl	Listening daemon-, receives and blocks IP-address of attacked host
	rain.pl	Script to remove old messages from system after a certain predefined time period
Herd size	deamon.pl	Scripts on hosts to collect machine load and send to collecting agent
	get_status.pl	Script to receive and average the received hosts load. Responsible for data consistency
	scale.pl	Scaling script to increase and decrease number of servers by the average load on all systems
Misc	finn.pl	Recreate load profile from finn.no using httperf
	K10SQS	Shutdown script to send host poweroff-message to collecting agent
	S10SQS	Boot script to send host online-message to collecting agent
	status.php	PHP-script to monitor the state of virtual machines status and load

Table 6.2: Table of scripts

Chapter 7

Conclusion

A journey of a thousand miles begins with a single step. Using biology as a platform to manage virtual machines has proved to be a step in the right direction, but has its limitations. Self-management in system administration is and will continue to be an important field of study, due to ever increasing system complexity and cost of administration.

By using biology it was possible to not only simplify complex technical solutions, but also use biology as a method to design new types of automated management processes. This processes have been translated into working examples and tested to demonstrate their effectiveness in re-created real-life scenarios.

An animals ability to adapt to change in environment, the ability to protect itself and the herd is a part of what makes animals so inspiring. By using biology it is possible to obtain ideas, discover strengths and weaknesses. So far the use of biology has been an untapped resource in system administration. If this thesis has proved something-, it is the potential for biology in creating automated management processes.

Chapter 8

Appendix

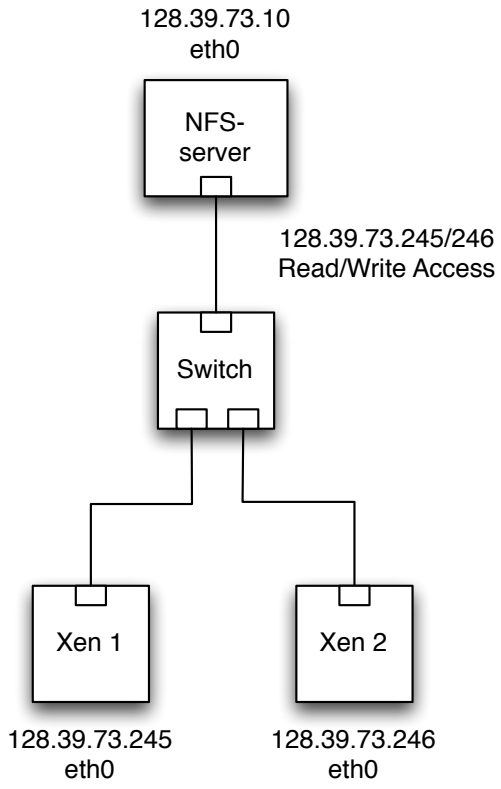


Figure 8.1: Physical server topology

8.1 Hardware and software specific

8.1.1 Hardware specifications

Dell Optiplex 745

CPU: Intel Core 2 CPU 6600 2.40GHz
MEM: 1 GB
STORAGE: 1 TB

Dell IBM x3550

Model: 7978B1G
CPU: Xeon Quad Core E5405
MEM: 4x1GB PC2-5300 CL5 ECC DDR2
STORAGE: 2x146 GB 15K 3,5 SAS

Amazon small instance

CPU: 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit)
MEM: 1.7 GB
STORAGE: 160 GB

8.1.2 Software specifications

Xen

Xen version: 2.6.32-5-xen-amd64
Plattform: x86_64

MLN

MLN version: 1.0.7
EC2 plugin version: 1.2

Amazon modules

Modules: modules-2.6.16-ec2
AMI version: ec2-ami-tools-1.3-66634
API version: api-tools-1.4.2.2
JAVA version: jre1.6.0_24

8.1.3 Crashreport Amazon

```
1  kernel:Oops: 0003 [#4]
2  Message from syslogd@webserver_4 at May 1 12:58:03 ...
3   kernel:SMP
4  Message from syslogd@webserver_4 at May 1 12:58:03 ...
5   kernel:CPU: 0
6  Message from syslogd@webserver_4 at May 1 12:58:03 ...
7   kernel:EIP is at pgd_free+0x104/0x140
8  Message from syslogd@webserver_4 at May 1 12:58:03 ...
9   kernel:eax: 00000000 ebx: 00000000 ecx: 00000400 edx: 80000004
10 Message from syslogd@webserver_4 at May 1 12:58:03 ...
11  kernel:esi: ec7de000 edi: ec7de000 ebp: ec785d68 esp: ec785d48
12 Message from syslogd@webserver_4 at May 1 12:58:03 ...
13  kernel:ds: 007b es: 007b ss: 0069
14 Message from syslogd@webserver_4 at May 1 12:58:03 ...
15  kernel:Process daemon.pl (pid: 4530, threadinfo=ec784000 task=ed6e1550)
16 Message from syslogd@webserver_4 at May 1 12:58:03 ...
17  kernel:Stack: <0>ec7de000 00000000 00000003 ec612628 ec9d8018 ec72f200 ↩
        ec72f248 ec72f200
18 Message from syslogd@webserver_4 at May 1 12:58:03 ...
19  kernel: ec785d7c c011bd9b ec9d8000 ec72f248 ec72f200 ec785d90 c011be50 ↩
        ec72f200
20 Message from syslogd@webserver_4 at May 1 12:58:03 ...
21  kernel: ec785de8 00007ff0 ec785e20 c0170a8a ec72f200 ec72f200 c030b547 ↩
        ec785db8
22 Message from syslogd@webserver_4 at May 1 12:58:03 ...
23  kernel:Call Trace:
24 Message from syslogd@webserver_4 at May 1 12:58:03 ...
25  kernel: [<c01058bd>] show_stack_log_lvl+0xcd/0x120
26 Message from syslogd@webserver_4 at May 1 12:58:03 ...
27  kernel: [<c0105abb>] show_registers+0x1ab/0x240
28 Message from syslogd@webserver_4 at May 1 12:58:03 ...
29  kernel: [<c0105dc1>] die+0x111/0x240
30 Message from syslogd@webserver_4 at May 1 12:58:03 ...
31  kernel: [<c01130a7>] do_page_fault+0x5f7/0x931
32 Message from syslogd@webserver_4 at May 1 12:58:03 ...
33  kernel: [<c01052ab>] error_code+0x2b/0x30
34 Message from syslogd@webserver_4 at May 1 12:58:03 ...
35  kernel: [<c011bd9b>] __mmdrop+0x1b/0x50
36 Message from syslogd@webserver_4 at May 1 12:58:03 ...
37  kernel: [<c011be50>] mmput+0x80/0xa0
38 Message from syslogd@webserver_4 at May 1 12:58:03 ...
39  kernel: [<c0170a8a>] flush_old_exec+0x1ba/0xb30
40 Message from syslogd@webserver_4 at May 1 12:58:03 ...
41  kernel: [<c0191e1f>] load_elf_binary+0x26f/0x1780
42 Message from syslogd@webserver_4 at May 1 12:58:03 ...
43  kernel: [<c0171602>] search_binary_handler+0x92/0x240
```

```
Message from syslogd@webserver_4 at May 1 12:58:03 ...
 kernel: [<c0171923>] do_execve+0x173/0x215
Message from syslogd@webserver_4 at May 1 12:58:03 ...
 kernel: [<c0103892>] sys_execve+0x42/0xa0
Message from syslogd@webserver_4 at May 1 12:58:03 ...
 kernel: [<c0105119>] syscall_call+0x7/0xb
Message from syslogd@webserver_4 at May 1 12:58:03 ...
 kernel:Code: e1 0c 25 ff 0f 00 00 09 c1 89 ce 83 ce 01 81 ee 01 00 00 40 31 db 89 5↩
      c 24 04 89 f7 89 34 24 e8 53 eb ff ff 31 c0 b9 00 04 00 00 <f3> ab a1 e4 ec 3a ↩
      c0 89 74 24 04 89 04 24 e8 39 dc 04 00 8b 45
```

8.2 Scripts

Scaling decision mechanism

```
if ( ($cycle_counter % 20) == 0 )
{
    print "Walking to the elephant graveyard\n";
    if ( $total_long_load < 20 )
    {
        print "Load lower than 15 % \n";
        $new_number = $up_count − int($up_count∗0.70);
        $new_number = ceil($new_number);
        if ( $new_number == 1 && $up_count > 1 && $up_count < 4)
        {
            $new_number = 2;
            print "New_number was 1, now is 2 due to number of hosts is ↩
                $up_count\n";
        }
        print "Decrease by 30 %. New−number: $new_number\n";
        scaledown($new_number);
    }

    else
    {
        $new_number = $up_count − int($up_count∗0.9);
        print "Decrease by 10 %. New−number: $new_number\n";
        scaledown($new_number);
    }
}

elsif ( ($cycle_counter % 10) == 0 )
{
    print "Time to reproduce\n";
    if ( $total_load < 40 && $total_load > 20 )
      {
```

```
31              $new_number = ceil(((($up_count*1.25) +1 ) -$up_count);
32              print "Increase by 25 %. New-number: $new_number\n";
33          }
34      elsif ( $total_load > 40 && $total_load < 100 )
35          {
36              $new_number = ceil(((($up_count*1.50 ) +1) -$up_count);
37              print "Increase by 50 %. New-number: $new_number\n";
38          }
39      else
40          {
41              $new_number = 1;
42              # Need to oppress need to scale to much: ceil(($up_count*1.10 )-↩
                    $up_count);
43              print "Increase by 1. New-number: $new_number\n";
44          }
45      $new_number = 1 if $new_number == 0;
46      scaleup("$new_number");
47  }
```

Scale-up mechanism

```
1   sub scaleup {
2       ### Run sanity check on number max/min
3       ### Find available machines
4       ### start machine(s)
5       ### Update queue-count of VMs
6       my $number = $_[0];
7       my $death_num = count_elements(@death_row);
8       my $preg_num = count_elements(@pregnant);
9       debug("Number: $number, Death_num: $death_num, Preg_num: ↩
            $preg_num\n");
10      if ( $death_num+$preg_num+$up_count >= $max_hosts )
11      {
12          print "No more room for more VMs\n";
13      }
14      else
15      {
16          my $avail_host = $max_hosts - $death_num - $preg_num - $up_count↩
                ;
17          print "Avaiable hosts: $avail_host\n";
18          $number = $avail_host if $avail_host < $number;
19          for ( my $i = 1; $i <= $number; $i++ ){
20              my $candidate_num;
21              my $candidate_name;
22              for ( my $k = 1; $k <= $max_hosts; $k++ ){
23                  my $name = get_name($k);
24
```

```
25                  debug("Checking host $name: dr: $death_row[$k], hh: ↩
                        $hash_host{$name}{'status'}, pr: $pregnant[$k]\n");
26                  if ( ( not $death_row[$k] ) and ( not $hash_host{$name}{"status↩
                        "} eq "up" ) and ( not $pregnant[$k]) ){
27                      verbose("We found a candidate!: $k\n");
28                      $candidate_num = $k;
29                      $candidate_name = $name;
30                      last;
31                      # start the VM!!!
32
33
34
35                  }
36
37
38              }
39              $pregnant[$candidate_num] = 1;
40
41              system("mln start -p $PROJECT -h $candidate_name &");
42              $nest{"$candidate_name"} = 0;
43          }
44      }
45
46  }
```

Scale-down mechanism

```
1   sub scaledown {
2       ### Run sanity check on number max/min
3       ### Find available machines
4       ### stop machine(s)
5       ### Update death-row of VMs
6       my $number = $_[0];
7       my $death_num = count_elements( @death_row);
8       my $preg_num = count_elements(@pregnant);
9       if ( $up_count > 1 )
10      {
11  # print "No more room for more VMs\n";
12
13          # adjust number of candidates
14          if ( $up_count == $number )
15          {
16              $number = $number - 1;
17          }
18          elsif ( $up_count < $number )
19          {
20              $number = $up_count;
```

```
21              $number = $number − 1;
22          }
23          debug("Entering first_for_loop in scaledown\n");
24          for ( my $i = 1; $i <= $number ; $i++ ){
25          ## find candidate for destruction (oldest one?):
26              debug("I: $i, number: $number\n");
27              my $max_age = 0;
28              my $max_age_candidate;
29              debug("Entering second_for_loop in scaledown\n");
30              for ( my $k = 1; $k <= $max_hosts; $k++ ){
31                  my $name = get_name($k);
32                  debug("K: $k, max_host: $max_hosts, name: $name\n");
33                  if ( not $pregnant[$k] and $hash_host{$name}{"status"} eq "up↩
                         " and not $death_row[$k] ){
34                      debug("found candidate $k ($name), is it old enough? ↩
                             $hash_host{$name}{'age'} \n");
35                      if ( $max_age == 0 )
36                      {
37                          $max_age_candidate = $k;
38                          $max_age = $hash_host{$name}{'age'};
39                      }
40                      elsif ( $max_age > $hash_host{$name}{'age'} )
41                      {
42                          $max_age_candidate = $k;
43                          $max_age = $hash_host{$name}{'age'};
44                      }
45                  }
46              }
47              verbose("Candidate is $max_age_candidate\n");
48              $death_row[$max_age_candidate] = 1;
49              my $name = get_name($max_age_candidate);
50
51              if ( $hash_host{"$name"}{'ip'} )
52              {
53                  my $delete_name = $hash_host{"$name"}{'ip'};
54                  remove($delete_name);
55                  verbose("Remove from pool first\n");
56              }
57              $lock_host{$name} = 1;
58              system("mln stop −p $PROJECT −h $name &");
59
60
61          }
62              }
63              else
64              {
65                  print "Spare him!\n";
66              }
67
```

```
68          }
```

Alert_daemon.pl

```
1  while ( 1 )
2      {
3          open(DEBUG,">>/tmp/alert.debug");
4          open(FILE,">>/tmp/snort.log");
5          my $alert;
6          my $alert_level = 0;
7          my $time = time();
8          open(SNORT,"/var/log/snort/alert") or die "Can't read snort alert-file $!\n";
9          while ( my $line = <SNORT> )
10         {
11             chomp $line;
12 # print "SNORT: $line\n";
13             if ( $line =~ /\[Classification: (.*)\] \[Priority: (\d+)\]/ )
14             {
15 # print $line;
16                 $alert = $1;
17                 $alert_level = $2;
18                 if ( $alert_level > 10 )
19                   {
20                       print "ALERT: $alert\n";
21                       print "Threatlevel: $alert_level\n";
22
23                       $p->SendMessage("ATTACK: $hostname ($ip) is attacked. $alert. Priority: $alert_level");
24                       $q->SendMessage("$ip");
25
26                       $p->SendMessage("ATTACK: $hostname ($ip) is shutting down");
27
28                       # Remove snort to prevent infinite boot
29                       system("rm /var/log/snort/alert/");
30                       system("poweroff");
31                       print "Send warning\n";
32                       print "Shutting down\n";
33                       print "Exiting script\n";
34                       exit 0;
35                   }
36             }
37         }
38
39         # Send a message
40         $q->SendMessage('Kicking ass and chewing bubblegum\n');
```

```
41
42          close SNORT;
43          print DEBUG "$time,sleeping\n";
44          close FILE;
45          close DEBUG;
46          sleep 5;
47
48      }
49  }
```

Listen_daemon.pl

```
1           # Enter loop to do work
2       for (my $i = 1;$i > 0;$i++)
3       {
4           open(DEBUG,">>/root/debug.log");
5           open(FILE,">>/root/localsyslog.log");
6
7           my $time = time();
8
9
10          # Send a message
11          # $q->SendMessage('Kicking ass and chewing bubblegum\n');
12
13          # Retrieve a message
14          $msg = $q->ReceiveMessage();
15          if ( $msg )
16          {
17              $message = $msg->MessageBody();
18  # print "Message IP: $message\n";
19
20              if ( $message =~ /(\d+.\d+.\d+.\d+)$/ )
21              {
22                  $message = $1;
23                  my $scalar = scalar(@array);
24                  if ( $scalar > 0 )
25                  {
26                      foreach my $element ( @array )
27                      {
28                          if ( $element =~ /$message/ )
29                          {
30                              print FILE "Already sent warning\n";
31                          }
32                          else
33                          {
34                              push(@array,$message);
```

```
35                                  system("iptables -A INPUT -s $message -j DROP");
36                                  print FILE "$time,blockip:$message\n";
37                                  $p->SendMessage("BLOCK:$hostname,$time,block_ip:$message");
38                              }
39                          }
40                      }
41                      else
42                      {
43                          push(@array,$message);
44                          system("iptables -A INPUT -s $message -j DROP");
45                          print FILE "$time,blockip:$message\n";
46                          $p->SendMessage("BLOCK:$hostname,$time,block_ip:$message");
47                      }
48                  }
49                      else
50                  {
51                      print FILE "$time,wrong data supplied: $message\n";
52                  }
53
54                  $q->ChangeMessageVisibility($msg->ReceiptHandle(),0);
55                  # Sets visibility to 0 to allow other hosts to read message
56                  ## Delete message
57              }
58              else
59              {
60                  print FILE "$time,null_messages\n";
61              }
62              print DEBUG "$time,sleeping\n";
63              close FILE;
64              close DEBUG;
65              sleep 5;
66
67      }
```

Status.php

```
1  <HTML>
2  <META HTTP-EQUIV="REFRESH" CONTENT="10">
3  <HEAD>
4      <LINK href="stylesheet.css" rel="stylesheet" type="text/css">
5    </HEAD>
6  <BODY>
7  <h1 class=header>Herd status</h1>
8
```

```
<table class=row>
<tr valign="top" ><td><table>

<?php
# phpinfo();
$file = $_GET['file'];
$contents = file("$file");
    echo "<tr>\n";
    echo "<td><b>Date</b></td><td></td>";
    for($h = 1; $h <= 8; $h++){
        echo "<td><b>Host $h</b></td>\n";
    }
echo "<td><b>Load</b></td>";
# echo $line . "<br>";
echo "</tr>";

$firstrun = 0;
$hostcount = 0;
$hostdata;
$inputdata;
$maxinput = 0;
$maxload = 0;
$hoststatus;
$curr_load = 0;
for ($i = count($contents) - 1; $i >=0 ; $i--) {
    $line = $contents[$i];
    $ar = split(",",$line);
    echo "<tr>\n";
    echo "<td>$ar[0]</td>";
    echo "<td>" . date("H:m:s",$ar[0]) . "</td>";
    for($h = 1; $h <= 8; $h++){

        # chart only
        if ( $i > (count($contents) - 52) ){
            if ( preg_match("/\d+/",$ar[$h]) ){
                $hostdata[$h] .= $ar[$h] . ",";
                if ( $ar[$h] > $maxload ){
                    $maxload = $ar[$h];
                }
            } else {
                $hostdata[$h] .= "0,";
            }
        }


        $style = "class=down";
        if( strcmp($ar[$h], "down") ){
            $style="class=up";
            if ( $firstrun == 0 ){
```

```
58                      $hostcount++;
59                      $hoststatus[$h] = 1;
60                  }
61
62              }
63              echo "<td $style >$ar[$h]</td>\n";
64
65
66          }
67          $style = "load0";
68          if ( $ar[9] > 0 and $ar[9] < 10 ){
69              $style = "load1";
70          } elseif ( $ar[9] >= 10 and $ar[9] < 20 ){
71              $style = "load10";
72          } elseif ( $ar[9] >= 20 and $ar[9] < 30 ){
73              $style = "load20";
74          } elseif ( $ar[9] >= 30 and $ar[9] < 40 ){
75              $style = "load30";
76          }
77          echo "<td class=$style >$ar[9]</td>\n";
78          if ( $i > (count($contents) - 52) ){
79              $ar[9] = trim($ar[9]);
80              $curr_load = $ar[9];
81  # echo "'$ar[9]'<br>";
82              if ( $ar[9] ){
83                  $inputdata .= $ar[9] . ",";
84              } else {
85                  $inputdata .= "0,";
86              }
87              if ( $maxinput < $ar[9] . "," ){
88                  $maxinput = $ar[9];
89              }
90          }
91          $firstrun = 1;
92  # echo $line . "<br>";
93          echo "</tr>";
94  }
95
96
97   echo "</table></td><td><table><tr><td class=box>\n";
98
99  echo "Total up: $hostcount";
100 # echo "<tr><td class=box>Current load: $curr_load </td></tr>";
101
102 echo "</td></tr>";
103 echo "<tr><td><b>Charts</b></td></tr>\n";
104 #echo "<tr><td>$inputdata</td></tr>\n";
105 echo "<tr><td style='padding-left:14px'><b>L</b>";
106 echo "<img src='http://$h.chart.apis.google.com/chart?\n";
```

```
107 echo "cht=lxy&\n";
108 echo "chs=450x65&\n";
109 # echo "chd=t:\n";
110 echo "chd=t:0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,
111 40,42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,80,82,
112 84,86,88,90,92,94,96,98,100|\n";
113 $inputdata = substr($inputdata, 0, strlen($inputdata)-1);
114 echo $inputdata;
115 echo "&\n";
116 #echo "chxt=x,y&\n";
117 echo "chxr=0,0,40&\n";
118 #echo "chxl=1:10|25|50|100&\n";
119 echo "chco=000000&\n";
120 echo "chds=0,100,0,40&\n";
121 echo "chls=2,1,0&\n";
122 echo "chxt=y&\n";
123 echo "chg=10,10&\n";
124 echo "'>";
125
126
127 echo "</td></tr>\n";
128
129 for($h = 1; $h <= 8; $h++){
130         $style = "down";
131         if ( $hoststatus[$h] == 1 ){
132           $style = "up";
133         }
134         echo "<tr valign='top' ><td valign='top' class=$style><b>$h</b>";
135         echo "<img src='http://$h.chart.apis.google.com/chart?\n";
136         echo "cht=lxy&\n";
137         echo "chs=450x65&\n";
138         echo "chd=t:0,2,4,6,8,10,12,14,16,18,20,22,24,
139 26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,
140 62,64,66,68,70,72,74,76,78,80,82,84,86,88,90,92,94,96,98,100|\n";
141         $hostdata[$h] = substr($hostdata[$h], 0, strlen($hostdata[$h])-1);
142         echo $hostdata[$h];
143     echo "&\n";
144     echo "chxt=y&\n";
145 # echo "chds=0,50,0,150&\n";
146         echo "chxr=1,0,50|1,0,100&\n";
147 # echo "chxl=1:10|25|50|100&\n";
148     echo "chg=10,10&\n";
149         if ( $hoststatus[$h] == 1 ){
150         echo "chts=00ff00,15&\n";
151         } else {
152         echo "chts=ff0000,15&\n";
153         }
154         echo "chco=0000ff&\n";
155         echo "chds=0,100&\n";
```

```
156             echo "chls=2,1,0&\n";
157             echo "'>";
158             echo "</td></tr>\n";
159         }
160
161 echo "</table></td></tr></table>";
162
163
164 ?>
165 </td></tr>
166 </table>
167 </BODY>
168 </HTML>
```

Bibliography

[1] An inefficient truth. PC World, 01 2007.

[2] Pseudocode standard, 03 2011.

[3] Amazon. Amazon simple queue service (amazon sqs).

[4] Amazon. Amazon statement following dropping of hosting of wikileaks.

[5] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing, Feb 2009.

[6] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles, pages 164–177, New York, NY, USA, 2003. ACM.

[7] Chris Barnard. Animal Behaviour - Mechanism, Development, Function and Evolution, volume 1. Pearson - Prentice Hall, 2004.

[8] J. Barr. Host Your Web Site In The Cloud: Amazon Web Services Made Easy Amazon EC2 Made Easy. 2010.

[9] Kyrre Begnum. Xen virtualization and multi-host management using mln. In AIMS '07: Proceedings of the 1st international conference on Autonomous Infrastructure, Management and Security, pages 229–229, Berlin, Heidelberg, 2007. Springer-Verlag.

[10] Kyrre Begnum. Mln - sourceforge, 02 2011.

[11] Kyrre Begnum, Nii Lartey, and Lu Xing. Cloud-oriented virtual machine management with mln. In Martin Jaatun, Gansen Zhao, and Chunming Rong, editors, Cloud Computing, volume 5931 of Lecture Notes in Computer Science, pages 266–277. Springer Berlin Heidelberg, 2009. 10.1007/978-3-642-10665-1_24.

[12] Kyrre Begnum, Nii Apleh Lartey, and Lu Xing. Cloud-oriented virtual machine management with mln. In Proceedings of the 1st International Conference on Cloud Computing, CloudCom '09, pages 266–277, Berlin, Heidelberg, 2009. Springer-Verlag.

[13] M. Burgess. Principles of network and system administration. John Wiley & Sons Inc, 2004.

[14] Mark Burgess. Computer immunology. In Proceedings of the 12th USENIX conference on System administration, pages 283–298, Berkeley, CA, USA, 1998. USENIX Association.

[15] Ricky W. Butler. What is formal methods?

[16] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2, NSDI'05, pages 273–286, Berkeley, CA, USA, 2005. USENIX Association.

[17] M. Clavel, F. DurÂ·n, S. Eker, P. Lincoln, N. MartÌ-Oliet, J. Meseguer, and J. F. Quesada. Maude: specification and programming in rewriting logic. Theoretical Computer Science, 285(2):187 – 243, 2002.

[18] Fred Cohen. Computer viruses : Theory and experiments. Computers Security, 6(1):22 – 35, 1987.

[19] Wolf Country. The wolf pack, 02 2011.

[20] Reinhold Kröger Dan Marinescu. State of the art in autonomic computing and virtualization, 2007.

[21] Charles Darwin. The Origin of Species. Gramercy, May 1995.

[22] Larry Dignan. Amazon's web services outage: End of cloud innocence?

[23] T. Dobzhansky. Nothing in biology makes sense except in the light of evolution. The American Biology Teacher, 35(3):125–129, 1973.

[24] Drugs.com. Epinephrine.

[25] R. Duit. On the role of analogies and metaphors in learning science. Science Education, 75(6):649–672, 1991.

[26] M Edmunds. Defence in Animals: A Survey of Anti-Predator Defences. Harlow, Essex NY: Longman, 1974.

[27] Jonathan Fahey. How your brain tells time, 10 2009.

[28] Jacques Ferber. Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1999.

[29] Stephanie Forrest, Steven A. Hofmeyr, and Anil Somayaji. Computer immunology. Commun. ACM, 40:88–96, October 1997.

[30] Apache Software Foundation. Apache http server.

[31] The Apache Foundation. Apache module modstatus.

[32] A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. IBM Systems Journal, 42(1):5 –18, 2003.

[33] Gartner. Server virtualization: From virtual machines to clouds. Technical report, Gartner, 2010.

[34] S.M. Glynn. The teaching with analogies model. Children's comprehension of text, pages 185–204, 1989.

[35] Google. Google charts tools, May 2011.

[36] T.F. Hansen. Evolution. By Douglas J Futuyma. Evolution. By Douglas J Futuyma. Sunderland (Massachusetts): Sinauer Associates. 89.95.$xv+603p; ill.; index. ISBN : 0-87893-187-2.2005., 2005.

[37] S. Heim. The resonant interface. HCI foundations for interaction design.

[38] John L. Hoogland. Aggression, ectoparasitism, and other possible costs of prairie dog. JSTOR, 1979.

[39] Svend Andreas Horgen. Webprogrammering i PHP, volume 3. Gyldendal akademisk, 2009.

[40] David Mosberger HP. Httperf - http://www.hpl.hp.com/research/linux/httperf/.

[41] http://news.netcraft.com/. March 2011 web server survey.

[42] K.C. Jones. Music piracy costs u.s. economy $12.5 billion, report reveals.

[43] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. Computer, 36:41–50, January 2003.

[44] Arjen Krap John Sechrest Kyrre Begnum, Karst Koymans. Using virtual machines in system administration education. 2004.

[45] Finn Labs. Webtraffic - day - date unspecified.

[46] G. Low and L. Cameron. Researching and applying metaphor. Cambridge Univ Pr, 1999.

[47] K. Matthias. Towards autonomic management in system administration. 2008.

[48] S.P.B. Medawar. An unsolved problem of biology. Published for the College by HK Lewis, 1952.

[49] D. Mosberger and T. Jin. httperf—a tool for measuring web server performance. ACM SIGMETRICS Performance Evaluation Review, 26(3):31–37, 1998.

[50] S. Murugesan. Harnessing green it: Principles and practices. IT Professional, 10(1):24 –33, jan.-feb. 2008.

[51] G. Reggio and R. Wieringa. Thirty one Problems in the Semantics of UML 1.3 Dynamics. In Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'99)–Workshop" Rigorous Modelling and Analysis of the UML: Challenges and Limitations. Citeseer, 1999.

[52] M. Roesch et al. Snort-lightweight intrusion detection for networks. In Proceedings of the 13th USENIX conference on System administration, pages 229–238. Seattle, Washington, 1999.

[53] A. Rydell and S. Sundberg. Piraterna: de svenska fildelarna som plundrade Hollywood. Ordfront, 2009.

[54] Jonny Schulz. Sys::statistics::linux.

[55] T.A. Sebeok. Coding in the evolution of signalling behavior. Behavioral Science, 7(4):430–442, 1962.

[56] SCOTT SHANE and ANDREW W. LEHREN. Leaked cables offer raw look at u.s. diplomacy.

[57] Søderberg. Allt Mitt är Ditt-Upphovsrätt, Fildelning Försörjning. 2008.

[58] N.E. Stork. How many species are there? Biodiversity and Conservation, 2(3):215–232, 1993.

[59] Unnamed. Bear climbing bird cage. http://www.quitor.com/bear.html.

[60] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. SIGCOMM Comput. Commun. Rev., 39:50–55, December 2008.

[61] Jennifer Viegas. Animals that play dead sacrifice others, 04 2009.

[62] Laurie J. Vitt, Justin D. Congdon, and Nancy A. Dickson. Adaptive strategies and energetics of tail autonomy in lizards. Ecology, 58(2):pp. 326–337, 1977.

[63] VMWare. Vmware vsphere dynamic resource balancing, 01 2010.

[64] Barbara Webb. Using robots to understand animal behavior. volume 38 of Advances in the Study of Behavior, chapter 1, pages 1–58. Academic Press, 2008.

[65] Simon Whitaker. Amazon-sqs-simple.